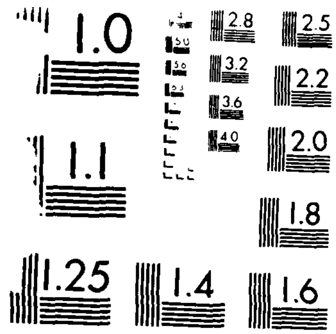
AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

1/6

UNCLASSIFIED F/G 5/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

1986

Ada® Training Curriculum

AD-A165 301

Software Engineering Methodologies
M201
Teacher's Guide
Volume II

Superseded
AD A144 257

U.S. Army Communications-Electronics Command
(CECOM)

Contract DAAB07-83-C-K506

Prepared By:

SOFTECH, INC.
460 Totten Pond Road
Waltham, MA 02154

INSTRUCTOR NOTES

<u>THEME</u>: THE SOFTWARE COST REDUCTION PROJECT METHODOLOGY (ANALYSIS PART) HOLDS GREAT PROMISE FOR DEVELOPING SPECIFICATIONS DURING THE ANALYSIS PHASE. IT IS BASED ON CONCEPTS OF:

- INFORMATION HIDING
- SEPARATION OF CONCERNS

<u>PURPOSE</u>: PROVIDE EXPOSURE TO KEY CONCEPT OF THE SPECIFICATION TECHNIQUES. EMPHASIS ON THE DOCUMENTATION STRUCTURE AND FORMATS.

<u>REFERENCES</u>:

1. K. HENNINGER, "SPECIFYING SOFTWARE REQUIREMENTS FOR COMPLEX SYSTEMS: NEW TECHNIQUES AND THEIR APPLICATION," IEEE TRANS. ON SOFTWARE ENGINEERING, VOL. SE-6, NO. 1, JANUARY 1980.

2. S. HESTER, D. PARNAS, D. UTTER, "USING DOCUMENTATION AS A SOFTWARE DESIGN MEDIUM," THE BELL SYSTEM TECHNICAL JOURNAL, VOL. 60, NO. 8, OCTOBER 1981.

Section 11
SOFTWARE COST REDUCTION PROJECT (SCRP) METHODOLOGY

Dist
A-1

VG 778.1

INSTRUCTOR NOTES

THE SOFTWARE COST REDUCTION PROJECT IS THE ONLY KNOWN ATTEMPT (OR EXPERIMENT) IN APPLYING A COORDINATED SET OF SOFTWARE ENGINEERING PRINCIPLES AND TECHNIQUES TO THE (RE)DEVELOPMENT OF A COMPLEX, REALISTICALLY SIZED, REAL TIME EMBEDDED COMPUTER APPLICATION - THE OPERATIONAL FLIGHT PROGRAM (OFP) OF THE A-7E AIRCRAFT. THIS WORK IS ALSO KNOWN AS THE A-7E PROJECT AND IS ALSO SOMETIMES REFERRED TO AS THE PARNAS METHODOLOGY.

THIS PROJECT COVERS THE COMPLETE SOFTWARE DEVELOPMENT LIFE-CYCLE FROM SOFTWARE REQUIREMENTS SPECIFICATION TO INTEGRATION AND TEST AND THE SUBSEQUENT MAINTENANCE ACTIVITY. THIS PRESENTATION FOCUSES ON THE REQUIREMENTS SPECIFICATION PROCESS - THE DESIGN ASPECTS OF THIS PROJECT WILL BE COVERED IN THE DESIGN PORTION OF THIS COURSE.

THE SCRP PROJECT IS AN OUTGROWTH OF THE WORK OF DAVID PARNAS AND OTHERS ON MODULARITY, INFORMATION HIDING, AND OTHER SOFTWARE ENGINEERING PRINCIPLES AND TECHNIQUES. DAVE PARNAS WORKED AT NRL DURING THE EARLY PHASES OF THIS PROJECT AND STILL SERVES AS A CONSULTANT EVEN THOUGH HE IS NOW AT THE UNIVERSITY OF VICTORIA. MANY TALENTED INDIVIDUALS HAVE CONTRIBUTED TO THIS WORK, BUT DAVE PARNAS IS CLEARLY THE SINGLE INDIVIDUAL MOST RESPONSIBLE FOR THESE IDEAS. THE NAVAL RESEARCH LAB (NRL) IS PRIMARILY THE LEAD ORGANIZATION, BUT SOME OF THE PARTICIPANTS (AND SPONSORSHIP) COME FROM THE NAVAL WEAPONS CENTER AT CHINA LAKE, CA. THE PROJECT WAS ONE OF THE ORIGINAL 10 STARS* PROJECTS FUNDED IN FY84 AND HAS MAINTAINED THAT STATUS THROUGH THE VARIOUS REDIRECTIONS OF THE STARS PROJECT.

THE PROJECT WAS STARTED IN RESPONSE TO THE RECOGNITION THAT THE MANY PROMISING SOFTWARE ENGINEERING TECHNIQUES BEING ESPOUSED (MOSTLY BY ACADEMICIANS) WERE NOT BEING PUT TO USE ON ANY "REAL" PROBLEMS. THE SOFTWARE ENGINEERS IN THE FIELD NEEDED A FULL-SCALE DEMONSTRATION THAT THE METHODS WOULD WORK ON A REAL TIME CRITICAL SYSTEM. THEY ALSO NEEDED A FULL WORKED OUT EXAMPLE TO USE OR A MODEL. (FOR BACKGROUND READ THE INTRO TO THE HENNINGER PAPER.)

*OVERVIEW OF STARS IS INCLUDED IN INTRODUCTORY SECTION OF THE COURSE.

SOFTWARE COST REDUCTION METHODOLOGY (SCRP)

BACKGROUND

- SOFTWARE COST REDUCTION PROJECT
  - NRL & NWC, STARTED - 1978
  - STARS FUNDED IN FY 84 & 85
- "THE PROBLEM"
  - NAVY SOFTWARE IS EXPENSIVE TO MAINTAIN
  - IMPROVED TECHNIQUES PROPOSED, BUT NOT USED
    - NO CONVINCING TEST
    - NO MODELS TO EMULATE
    - FEAR OF PERFORMANCE IMPACT

INSTRUCTOR NOTES

THE SCRP PROJECT HAS APPLIED MODERN SOFTWARE ENGINEERING TECHNIQUES (WHICH ARE DISCUSSED LATER) TO A REAL PROBLEM - THE OFP OF THE A-7E AIRCRAFT. THE PROJECT IS BEING CLOSELY MONITORED SO THAT THE "COST" OF THESE TECHNIQUES CAN BE DEMONSTRATED. THE PROJECTS WILL CONTINUE TO COLLECT EXTENSIVE DATA ON THE COST OF MAINTAINING THE REDEVELOPED SOFTWARE SYSTEM (BOTH IT AND THE ORIGINAL SYSTEM, DONE IN THE TRADITIONAL AD-HOC METHOD - ASSEMBLY LANGUAGE - WILL BE MAINTAINED IN PARALLEL).

THE PROJECT IS ALSO PRODUCING "MODEL DOCUMENTS" WHICH CONTAIN TUTORIAL INFORMATION WHICH HAS ALLOWED OTHER PROJECTS TO APPLY THE TECHNIQUES BY STUDYING THE COMPLETE, WORKED OUT EXAMPLE. SEVERAL TUTORIAL PAPERS HAVE ALSO BEEN PRODUCED.

BACKGROUND - continued

- "THE SOLUTION" - A SOFTWARE DEVELOPMENT METHODOLOGY
  - APPLY IMPROVED TECHNIQUES TO AN EXISTING SYSTEM
  - PROVIDE CONVINCING TEST WITH NAVY SYSTEM
  - PROVIDE MODELS THAT CAN BE EMULATED
    - MODEL DOCUMENTS
    - MODEL DESIGN
    - MODEL CODE
  - PROVIDE TUTORIAL MATERIAL TO TRANSFER THE TECHNOLOGY

INSTRUCTOR NOTES

THE COLLECTING OF MEASUREMENT DATA IS INTENDED TO DEMONSTRATE THE COST-BENEFIT RATIO OF APPLYING THESE TECHNIQUES. IN PARTICULAR THE EXECUTION TIME PENALTIES FOR USING THE MODULARITY AND SYNCHRONIZATION APPROACHES, AS WELL AS A HIGH LEVEL LANGUAGE WILL BE MEASURED. THE EXISTING A-7E OFP IS A TIME CRITICAL PROGRAM WHICH "JUST FITS" IN THE CURRENT TC-2 COMPUTER (BOTH TIME AND SPACE WISE). ANY OVERHEAD ASSOCIATED WITH THE NEW TECHNIQUES WILL BE READILY APPARENT. AT THE SAME TIME THE SWINGS IN MAINTENANCE COSTS HAVE ALREADY BEGUN TO BE DEMONSTRATED. BOTH THE NAVY AND THE AIR FORCE (WHICH USES A SLIGHTLY MODIFIED VERSION) HAVE BEGUN USING THE SOFTWARE REQUIREMENTS SPEC BECAUSE IT IS SO EASY TO UNDERSTAND AND TO CHANGE.

IN ADDITION TO THE CONTINUED USE BY NRC ON THE A-7E AND OTHER (CLASSIFIED) PROJECTS, MANY COMPANIES HAVE SUCCESSFULLY TRANSITIONED THE TECHNOLOGY. SOFTECH, GRUMMAN, BELL LABS AND OTHER COMPANIES HAVE APPLIED AND ADOPTED THE METHODOLOGY TO A WIDE CLASS OF APPLICATIONS.

BACKGROUND - continued

- "EXPECTED RESULTS"
  - DO TECHNIQUES AND PRINCIPLES WORK WITH EXTERNAL CONSTRAINTS?
  - WHAT ARE THE COSTS IN COMPUTATIONAL RESOURCES?
  - HOW DO LITTLE-SYSTEM PRINCIPLES SCALE UP?
  - *IS THE SYSTEM MEASURABLY BETTER?*

- CURRENT USAGES
  - NRL WORK CONTINUING WITH THE PARALLEL DEVELOPMENT AND MAINTENANCE OF THE OPERATIONAL FLIGHT PROGRAM OF THE A-7E
  - USED/EXTENDED ON NUMEROUS BELL LAB PROJECTS
  - GRUMMAN, ...

INSTRUCTOR NOTES

(READ SECTION IV OF THE HENNINGER PAPER.)

EACH OF THE DOCUMENTS ARE ORGANIZED INTO SECTIONS WHICH CONTAIN LOGICALLY SEPARATE TYPES OF INFORMATION, DETAILS WHICH ARE LIKELY TO CHANGE SEPARATELY AS CHANGES ARE MADE TO THE SYSTEM. WITHIN EACH SECTION, A PARTICULAR TEMPLATE FOR THE REQUIRED INFORMATION IS USED. THIS TEMPLATE INSURES THAT ALL THE RIGHT QUESTIONS ARE ASKED (AND THEN ANSWERED). THE ANSWERS ARE PROVIDED USING WELL DEFINED CONSISTENT NOTATION (OR MODELS AND TECHNIQUES) SO THAT THE DOCUMENT IS UNIFORM AND VERIFIABLE.

MAIN PRINCIPLES OF THE SCRP METHODS

- ASK THE RIGHT QUESTIONS AT THE RIGHT TIME
- BUILD ON THE EXPERIENCE OF OTHERS
- USE A DOCUMENT FRAMEWORK TO ORGANIZE ANSWERS TO ALL QUESTIONS
- WITHIN THE DOCUMENTS USE ONLY APPROPRIATE MODELS OR TECHNIQUES

INSTRUCTOR NOTES

THE USE OF STANDARD TEMPLATES AND THE ORGANIZATION OF INFORMATION IN THE DOCUMENTS IS INTENDED TO MAKE THE DOCUMENT PRECISE, AND AN EXPLICIT AGREEMENT BETWEEN THE USER OF THE SYSTEM, THE PROCURING AGENCY, AND THE DEVELOPER. IT IS ALSO INTENDED TO REDUCE THE MAINTENANCE COST FOR THE SYSTEM BY LIMITING THE IMPACT OF ANY CHANGE TO THE SYSTEM. YOU SHOULD ALWAYS KNOW WHEN SUCH A CHANGE SHOULD GO AND A SINGLE CHANGE TO THE SYSTEM SHOULD ONLY REQUIRE A CHANGE TO ONE PART OF THE DOCUMENT.

TEMPLATES ARE ALSO A MECHANISM FOR ORGANIZING THE WORK TO BE DONE IN PRODUCING A DOCUMENT - IT'S A "FILL IN THE BLANKS" SPECIFICATION PROCESS. THE PROCESS IS WELL SUITED TO AUTOMATION.

EXPECTED BENEFITS OF THE SCRP METHODS

- EASE OF CHANGE OF SYSTEM FUNCTIONS AND SOFTWARE IMPLEMENTATION
- CONTROL OF THE INFORMATION ABOUT THE SYSTEM AND IT'S IMPLEMENTATION
- ORDERING OF THE DEVELOPMENT STEPS TO MEET A PROJECT'S OBJECTIVES
- MAKING THE AGREEMENTS BETWEEN PROJECT PARTICIPANTS EXPLICIT
- LIMITATION OF THE SCOPE OF EFFECT OF FUNCTIONAL CHANGES ON THE DOCUMENTATION AND CODE

INSTRUCTOR NOTES

SEPARATION OF CONCERNS ATTEMPTS TO PARTITION INFORMATION SO THAT THE RIPPLE EFFECT OF CHANGES IS MINIMIZED. WE USE CATEGORIES OF INFORMATION THAT WILL CHANGE INDEPENDENTLY. FOR INSTANCE WE DESCRIBE THE MAPPING DETAILS OF A HARDWARE INTERFACE SEPARATE FROM THE FUNCTIONAL USE OF THE INFORMATION PROVIDED BY THAT INTERFACE. IF WE CHANGE TO A NEW DEVICE WE ONLY NEED TO CHANGE THE MAPPING. IF WE CHANGE THE FUNCTIONING OF THE SOFTWARE THEN WE DON'T NEED TO CHANGE THE HARDWARE MAPPING DESCRIPTIONS.

MANY EXAMPLES OF THE FORMALISMS USED IN THIS APPROACH WILL BE PRESENTED IN THE REST OF THIS SECTION. THE APPROACH IS FORMAL WITHOUT BEING UN-INTUITIVE - ITS A LEVEL OF FORMALISM THAT ENGINEERS ARE COMFORTABLE WITH.

ABSTRACT INTERFACE AND INFORMATION HIDING ARE PRINCIPLES APPLIED TO THE DESIGN PROCESS AND WILL BE COVERED AT A LATER TIME. ALL OF THE DOCUMENTS HAVE BEEN DESIGNED TO INSURE THAT ALL THE ANALYSIS AND DESIGN INFORMATION IS CAPTURED IN THE APPROPRIATE ORDER. (READ THE INTROS TO THE BSTJ ARTICLE AND THE HENNINGER PAPER.)

UNDERLYING CONCEPTS AND TECHNIQUES

- SEPARATION OF CONCERNS
- FORMAL SPECIFICATION
- ABSTRACT INTERFACES/INFORMATION HIDING
- DOCUMENTATION AS A SOFTWARE DESIGN MEDIUM*

*HESTER, PARNAS AND UTTER - BELL SYSTEM TECHNICAL JOURNAL, OCT. 81

INSTRUCTOR NOTES

(SEE PREVIOUS DISCUSSION) - THIS IS A VERY IMPORTANT PRINCIPLE - MOST CURRENT DOCUMENTS INTERMIX CONCERNS. ANY CHANGE TO THE SYSTEM HAS A RIPPLE EFFECT THROUGH THE DOCUMENTS. THIS IS WHERE WE ARE WASTING MOST OF OUR SOFTWARE DOLLARS. THIS PRINCIPLE APPLIES TO ALL DOCUMENTS AND TO THE CODE AS WELL.

AT A HIGH LEVEL, WE ORGANIZE THE DOCUMENT SO THAT INTERFACE (INPUT/OUTPUT MAPPING) INFORMATION IS SEPARATED FROM FUNCTIONAL, WHICH IS SEPARATED FROM TIMING, WHICH IS SEPARATED FROM ACCURACY.

AT A LOWER LEVEL WITHIN EACH OF THESE CATEGORIES WE USE TEMPLATES TO PROVIDE A STANDARD ORGANIZATION TO THE DETAILED INFORMATION.

SEPARATION OF CONCERNS

- GOALS -
  1. REDUCE THE IMPACT OF CHANGE
  2. ALWAYS KNOW WHERE TO FIND ANSWERS TO QUESTIONS

- SEPARATE
  (FOR A REQUIREMENTS SPECIFICATION)
  - FUNCTIONALITY FROM BEHAVIOR
  - INPUT/OUTPUT MAPPING FROM LOGICAL VALUES
  - TIMING
  - ACCURACY
  - EXPECTED CHANGES
  - RESPONSE TO UNEXPECTED CONDITIONS FROM NORMAL OPERATION

- USE STANDARD TEMPLATES FOR FURTHER SEPARATION AND TO FOCUS ON THE KEY QUESTIONS THAT SHOULD BE ANSWERED.

INSTRUCTOR NOTES

THE LEVEL OF FORMALISM IS A COMPROMISE BETWEEN A RIGID, NON-INTUITIVE APPROACH SUCH AS MATHEMATICAL FORMALISM OR A RIGID PROGRAMMING LANGUAGE AND COMPLETELY AMBIGUOUS ENGLISH LANGUAGE. MOST USERS ARE PUT OFF BY SOME OF THE SYMBOLS BUT IT GROWS ON YOU AFTER YOU WORK WITH IT. THE VARIOUS FORMALISMS ALLOW FOR EASY EFFECTIVE REVIEW AND FOR MACHINE SCANNING. FORMALISMS (AND ABBREVIATIONS) REDUCE THE BULK OF THE DOCUMENT.

(ALMOST) FORMAL SPECIFICATIONS

- BRIEF, PRECISE, AND CONCISE - AVOID REDUNDANCY
  - //OUTPUT//, /INPUT/
  - *MODE*
  - !TEXT MACRO!
  - +DATA TYPE+
- STANDARD TEMPLATES AND TABLES
- LOGICAL OPERATORS
- CONDITIONS AND EVENTS
- AVOID PROSE AS APPROPRIATE
- FORMAL PROCESSING OF SPECIFICATION NOT NECESSARY TO DETERMINE CONSISTENCY, ETC.
  - STANDARD TEXTUALLY ORIENTED TOOLS (GREP, ETC.) WILL WORK

INSTRUCTOR NOTES

THIS NOTATION IS EXPLAINED IN HENNINGER. MOST USER'S OF THE METHODOLOGY HAVE HAD TO INTRODUCE EXTENSIONS TO THE NOTATION TO HANDLE IDIOSYNCRASIES OF THE SPECIFIC SYSTEM THE NOTATION FOR SHARED DISPLAY DEVICE OUTPUT WAS ADDED BY SOFTECH TO HANDLE THE COMPLEX GRAPHICS INTERFACE OF THE EP SYSTEM.

NOTATION DESCRIPTION

- <u>NAMING CONVENTIONS</u>

/INPUT/	-	A SYSTEM LEVEL INPUT (TYPICALLY HARDWARE)
//OUTPUT//	-	A SYSTEM LEVEL HARDWARE OUTPUT
/(OUTPUT)/	-	A SHARED DISPLAY DEVICE OUTPUT
$VALUE$	-	A MNEMONIC NAME FOR A NON-NUMERIC VALUE
!TERM!	-	A COMPLEX OR OFTEN USED EXPRESSION OR TERM THAT IS FULLY DEFINED IN A DICTIONARY (TEXT MARCO DICTIONARY)
+DATA TYPE+	-	A DATA TYPE
MODE	-	AN OPERATIONAL MODE OR STATE OF THE SYSTEM

- <u>OPERATORS</u>

$<, \leq, >, \geq, =$	-	RELATIONAL OPERATORS
ABS, MAX, MIN, SIGN	-	ARITHMETIC FUNCTIONS
AND, OR, NOT	-	LOGICAL OPERATOR
+, _, *, /, ETC	-	NORMAL ARITHMETIC OPERATORS

INSTRUCTOR NOTES

THE CONCEPT OF INFORMATION HIDING AND THE USE OF ABSTRACT INTERFACES FOR DOCUMENTING MODULES HAS BEEN AROUND FOR A LONG TIME. PARNAS INTRODUCED THESE CONCEPTS OVER 10 YEARS AGO. THESE ARE SOMEWHAT HARD TO REALLY UNDERSTAND UNTIL YOU HAVE A GOOD WORKED OUT EXAMPLE. THE SCRP PROJECT HAS PROVIDED A LARGE NUMBER OF WELL DOCUMENTED EXAMPLES OF THESE TECHNIQUES.

ESSENTIALLY AN ABSTRACT INTERFACE ONLY *SHOWS A USER WHAT HE ABSOLUTELY NEEDS TO KNOW TO* USE A MODULE. JUST AS A ROAD MAP TELLS A DRIVER WHAT HE NEEDS TO KNOW. ROAD MAPS DON'T SHOW ELEVATION, OR TREES OR OTHER NON-ESSENTIAL INFORMATION THAT MAY CHANGE.

KEY PARTITIONING CRITERIA IS ALWAYS ON THE BASIS OF MINIMIZING THE IMPACT OF CHANGE. THIS MEANS THE DESIGNER MUST BE FAMILIAR WITH THE SYSTEM AND KNOW WHAT IS LIKELY TO CHANGE AND WHAT WILL NEVER CHANGE.

ABSTRACT INTERFACES/INFORMATION HIDING

- PARTITION SYSTEM ON THE BASIS OF EXPECTED CHANGE
- HIDE EFFECTS OF CHANGE IN SEPARATE MODULES
- ABSTRACT INTERFACE ONLY SHOWS UNCHANGING ASPECTS OF THE INTERFACE
- ABSTRACT INTERFACE IS TO MODULE, AS ROAD MAP IS TO WORLD
- THIS WILL BE COVERED IN DETAIL IN DESIGN SECTION OF THE COURSE

INSTRUCTOR NOTES

DOCUMENTS ARE ORGANIZED TO REFLECT THE ORDER IN WHICH DECISIONS MUST BE MADE AND TO RECORD ALL INFORMATION NEEDED TO IMPLEMENT AND MAINTAIN THE SOFTWARE. (READ INTRO TO BSTJ PAPER BY HESTER, PARNAS, AND UTTER.)

NOT ONLY ARE FINAL DECISIONS DOCUMENTED. ASSUMPTIONS AND SIGNIFICANT TRADE-OFFS ARE ALSO RECORDED TO ASSIST IN THE LIFE-CYCLE MAINTENANCE.

VARIOUS PHYSICAL PACKAGING APPROACHES HAVE BEEN USED. STUDIES HAVE BEEN CONDUCTED TO SHOW HOW THE PACKAGING CAN BE MADE COMPATIBLE WITH EXISTING MIL-STDS (SUCH REPACKAGING DOESN'T IMPROVE QUALITY, BUT IS USABLE).

DOCUMENTATION AS A SOFTWARE DESIGN MEDIUM

- PROVIDE A DOCUMENT FRAMEWORK OF QUESTIONS TO BE ANSWERED AND A PLACE TO PUT THE ANSWERS

- EXPLICITLY DOCUMENT ASSUMPTIONS

- DOCUMENTS BASED UPON SEPARATION OF CONCERNS AND INFORMATION HIDING CONCEPTS

- LOGICAL NATURE OF DOCUMENTS ARE SPECIFIED, PHYSICAL PACKAGING WILL BE PROJECT SPECIFIC

  - MIL-STD'S
  - CORPORATE/ORGANIZATIONAL REQUIREMENTS

INSTRUCTOR NOTES

THESE DEFINITIONS ARE OUT OF THE BSTJ PAPER. THE BELL WORK ADDED THE CONCEPT OF PROCESS. THIS ISN'T FORMALLY USED IN THE NRL PROJECT.

FUNCTIONS ARE ASSOCIATED WITH THE REQUIREMENTS SPECIFICATION - MODULE, ACCESS ROUTINES, AND PROCESS ARE ALL DESIGN TERMS.

SOME NEEDED DEFINITIONS

- FUNCTION - THE ALGORITHMS, RULES OR RELATIONSHIPS APPLIED BY THE SYSTEM IN RESPONSE TO EVENTS IN ORDER TO DETERMINE THE VALUES OF ONE OR MORE OUTPUT DATA ITEMS.

- MODULE - A PIECE OF SOFTWARE AND THE ASSOCIATED DOCUMENTATION WHICH TOGETHER CONTAIN ALL THE INFORMATION ABOUT SOME FUNCTION(S) OR A PART OF A FUNCTION. SMALL ENOUGH TO BE DEVELOPED BY ONE PERSON IN A LIMITED AMOUNT OF TIME (1-3 MONTHS).

- ACCESS ROUTINE - A PIECE OF SOFTWARE IN A MODULE WHICH CAN BE INVOKED BY SOFTWARE IN OTHER MODULES TO PERFORM SOME PORTION OF THE MODULE'S FUNCTION.

- PROCESS - A SET OF ACCESS ROUTINES WHOSE EXECUTION SEQUENCE IS PRESCRIBED. EXECUTION OF PROCESSES CAN BE CONCURRENT.

INSTRUCTOR NOTES

THE REQ SPEC IS A BLACK BOX VIEW OF THE SYSTEM AND ALL THE EXPLICIT DESIGN CONSTRAINTS THAT MUST BE IMPOSED.

THE SW MODULE GUIDE PROVIDES A HIERARCHICAL, THREE LEVEL DECOMPOSITION OF THE SYSTEM INTO MANY SMALL MODULES. THE TOP TWO LEVELS OF THIS DECOMPOSITION ARE PROBABLY APPLICABLE TO A WIDE RANGE OF EMBEDDED APPLICATIONS. THE SW MODULE GUIDE IS MEANT TO BE USED TO FIND OUT WHERE IN THE DESIGN STRUCTURE VARIOUS FUNCTIONAL REQUIREMENTS HAVE BEEN ALLOCATED.

FUNCTION DRIVER MODULE SPEC IS CLOSELY DERIVED FROM THE REQUIREMENTS SPEC. IT DESCRIBES THE STIMULUS RESPONSE BEHAVIOR OR THE SYSTEM IN TERMS OF CALLS ON ACCESS ROUTINES DEFINED IN THE ABSTRACT MODULE SPECS.

THE ABSTRACT MODULE INTERFACE SPECS WILL BE EXAMINED IN DETAIL IN THE DESIGN SECTION.

SCOPE OF DOCUMENTS OF THE SCRP METHODS
(AS APPLIED BY NRL)

ANALYSIS

- REQUIREMENTS SPECIFICATION - EVERYTHING THE SOFTWARE DESIGNERS NEED TO KNOW ABOUT THE SYSTEM.

DESIGN

- SOFTWARE MODULE GUIDE - THE HIERARCHICAL DECOMPOSITION OF THE SYSTEM INTO MODULES - ALLOCATES SYSTEM FUNCTIONS TO MANY SMALL MODULES.
- FUNCTION DRIVER MODULE SPECIFICATION - SPECIFIES THE STEP-BY-STEP EXECUTION TIME BEHAVIOR OF THE SYSTEM IN RESPONSE TO OPERATOR INPUTS
- ABSTRACT MODULE INTERFACE SPECIFICATIONS - EVERYTHING ANOTHER PROGRAMMER NEEDS TO KNOW TO CORRECTLY USE FUNCTIONS PROVIDED BY THE MODULE.

INSTRUCTOR NOTES

THE BELL LABS WORK MAPS CLOSELY TO THE NRL WORK. THE MODULE DECOMPOSITION DOCUMENT IS EQUIVALENT TO THE SW MODULE GUIDE. THE PROCESS STRUCTURE IS ESSENTIALLY EQUIVALENT TO THE FUNCTION DRIVER SPEC, BUT USES A RIGOROUS DEFINITION OF PROCESS. THE MODULE INTERFACE DOCUMENTS ARE EQUIVALENT TO THE ABSTRACT MODULE INTERFACE SPECS.

NRL DOESN'T HAVE ANYTHING EQUIVALENT TO THE MODULE DESIGN - THE FUNCTION D SAVER MODULE SPEC IS VERY DETAILED. NRL USES A VERY HIGH LEVEL LANGUAGE THAT IS CONSIDERED TO BE EQUIVALENT TO A DETAILED DESIGN DESCRIPTION.

NRL DOESN'T HAVE ANYTHING EQUIVALENT TO THE MODULE DEPENDENCY OR RESOURCE ALLOCATIONS ALTHOUGH SOME OF THE NRL CROSS INDICES PROVIDE SIMILAR INFORMATION.

SCOPE OF DOCUMENTS OF THE SCRP METHODS

(AS APPLIED BY BELL LABS)

ANALYSIS

- REQUIREMENTS SPECIFICATION - EVERYTHING THE SOFTWARE DESIGNERS NEED TO KNOW ABOUT THE SYSTEM

DESIGN

- MODULE DECOMPOSITION - THE DIVISION OF THE SYSTEM INTO MODULES
- MODULE DEPENDENCY - TABULATION OF THE OTHER MODULES WHICH EACH MODULE USES TO PERFORM ITS FUNCTIONS
- PROCESS STRUCTURE - GROUPING OF ACCESS ROUTINES THAT HAVE PRESCRIBED EXECUTION SEQUENCES
- RESOURCE ALLOCATION - SYSTEM RESOURCES USED BY EACH MODULE
- MODULE INTERFACE - EVERYTHING ANOTHER PROGRAMMER NEEDS TO KNOW TO CORRECTLY USE THE FUNCTIONS PROVIDED BY THE MODULE
- MODULE DESIGN - DESCRIPTION OF THE INTERNAL DESIGN OF A MODULE

INSTRUCTOR NOTES

OBJECTIVE OF THE REQ SPEC IS TO PROVIDE AN EXTERNAL, BL'CK-BOX DESCRIPTION IN AN UNAMBIGUOUS VERIFIABLE FORMAT. MANY EASY REFERENCE AIDS ARE PROVIDED, SUCH AS DICTIONARIES AND VARIOUS INDICES.

ANY DESIGN CONSTRAINTS ARE CLEARLY IDENTIFIED AND STATED SEPARATELY FROM FUNCTIONAL REQUIREMENTS.

REQUIREMENTS SPECIFICATION DOCUMENTS AND TECHNIQUES

- GOALS:
  - UNAMBIGUOUS
  - IMPLEMENTATION INDEPENDENT
  - EXTERNAL (BLACK-BOX) VIEW
  - EASY REFERENCE
  - COMPLETE
  - REVIEWABLE (VERIFIABLE)

INSTRUCTOR NOTES

DOCUMENT HAS MANY MORE SECTIONS THAN TRADITION PERFORMANCE SPECIFICATIONS - AN EXAMPLE OF SEPARATION OF CONCERNS.

MAJOR PORTIONS OF DOCUMENTS ARE SECTIONS 2, 3, 4, AND 10.

REQUIRED SUBSETS AND EXPECTED TYPES OF CHANGES ARE EXAMPLES OF DESIGN CONSTRAINTS AND *FORETHOUGHT THAT THE DESIGNER* NEEDS IN ADDITION TO THE FUNCTIONAL REQUIREMENTS.

COCUMENT CONTENT

0. INTRODUCTION
1. DISTINGUISHING CHARACTERISTICS OF THE COMPUTER ENVIRONMENT
2. INPUT AND OUTPUT DATA ITEMS
3. MODES OF OPERATION
4. TIME - INDEPENDENT DESCRIPTION OF FUNCTIONS
5. TIMING REQUIREMENTS
6. ACCURACY CONSTRAINTS ON FUNCTIONS
7. UNDESIRED EVENT RESPONSES
8. REQUIRED SUBSETS
9. EXPECTED TYPES OF CHANGES
10. GLOSSARY OF ABBREVIATIONS, ACRONYMS, TECHNICAL TERMS, INDICES, DICTIONARY
11. SOURCES OF FURTHER INFORMATION

OPTIONAL SECTION

- COMMUNICATION PROTOCOLS - SPECIFIES EXTERNAL PROTOCOL REQUIREMENTS

INSTRUCTOR NOTES

THESE FOUR SECTIONS ARE THE MOST SIGNIFICANT AND UNUSUAL ASPECTS OF THE REQUIREMENTS SPEC AND WILL BE DESCRIBED IN MORE DETAIL.

KEY SPECIFICATION COMPONENTS

- DATA ITEM DESCRIPTIONS
- TEXT MACROS
- FUNCTIONS
- MODES

INSTRUCTOR NOTES

READ SECTION V OF HENNINGER PAPER. DATA ITEM DESCRIPTIONS EXEMPLIFY TWO ASPECTS OF SEPARATION OF CONCERNS. AT A HIGH LEVEL WE HAVE SEPARATED ALL INPUT/OUTPUT PROCESSING OR MAPPING FROM THE USE OF THESE DATA ITEMS. WITHIN EACH DESCRIPTION WE DISTINGUISH BETWEEN THE ESSENTIAL CHARACTERISTICS - THOSE THINGS THAT ARE UNLIKELY TO CHANGE EVEN IF WE CHANGE A SENSOR - FROM THE ARBITRARY DETAILS, SUCH AS THE BIT MAPPING OR VALUE RESOLUTION.

IMPORTANT TO STRESS 1 TO N RELATIONSHIPS BETWEEN FUNCTIONS AND OUTPUTS - AN OUTPUT IS NEVER SET/MODIFIED/WHATEVER BY MORE THAN ONE FUNCTION.

NO SUCH RELATIONSHIP FOR INPUTS. INPUTS ARE TREATED AS RESOURCES. ANY FUNCTION CAN USE ANY (OR ALL) AVAILABLE INPUTS.

DATA ITEM DESCRIPTIONS

- SEPARATE ESSENTIAL CHARACTERISTICS FROM ARBITRARY DETAILS
- INPUT AND OUTPUT DATA ITEMS DESCRIBED SEPARATE FROM USE
- INPUTS TREATED AS RESOURCES
- INPUTS REPRESENT A MAPPING FROM EXTERNAL ACTION (SW CLOSURE) TO ARBITRARY DETAIL (BIT PATTERN) TO ESSENTIAL CHARACTERISTIC (ON, OFF)
- OUTPUTS REPRESENT A MAPPING FROM ESSENTIAL CHARACTERISTIC ($HIGH$, MED, LOW, OFF) TO ARBITRARY DETAIL TO THE VISIBLE EFFECT ON ASSOCIATED DEVICE
- 1 TO N RELATIONSHIP BETWEEN FUNCTIONS AND OUTPUT DATA ITEMS

INSTRUCTOR NOTES

THE TEMPLATE SHOWN NEXT IS ASSOCIATED WITH A LIST OF QUESTIONS THAT AIDS IN FILLING OUT THE TEMPLATE AND MAKING SURE ALL NECESSARY INFORMATION IS PROVIDED. SYMBOLIC NAMES FOR DATA TYPES AND VALUES SEPARATES THE FUNCTION DESCRIPTIONS FROM THE BIT LEVEL DETAILS OF THE DATA ITEM.

OUTPUTS ARE DESCRIBED IN TERMS OF VISIBLE EFFECTS - MAKES SYSTEM TESTABLE FROM OBSERVATION OF EXTERNAL BEHAVIOR.

DATA TYPE ADDED BY BELL LABS AND USED IN NRL DESIGN DOCUMENTS.

DATA ITEM DESCRIPTION TECHNIQUES

- USE A CONSISTENT APPROACH BASED ON KEY QUESTIONS TO BE ANSWERED
  - HOW IS THE PROGRAM TO READ/WRITE THE DATA?
  - WHAT IS THE BIT REPRESENTATION?
  - WHAT IS ALLOWED RANGE OF VALUES?
  - CAN COMPUTER TELL IF VALUE (OF SENSOR) IS VALID?
- USE SYMBOLIC NAMES FOR DATA TYPES, DATA ITEMS AND VALUES
- USE TEMPLATES FOR VALUE DESCRIPTIONS
- DESCRIBE OUTPUTS IN TERMS OF EFFECT ON EXTERNAL (HARDWARE) SYSTEM
- ALL DATA ITEMS MUST BE OF A VALID DEFINED DATA TYPE
  - ALLOWED DATA TYPES DEFINED IN TABLE INCLUDED INPUT/OUTPUT DATA ITEMS SECTION

INSTRUCTOR NOTES

POINTS TO BE MADE:

1. ACRONYM IS USED THROUGHOUT REMAINDER OF DOCUMENT AS ARE THE SYMBOLIC VALUES OF THE VALUE ENCODING.

2. CHARACTERISTICS OF VALUES SHOWS THE MAPPING BETWEEN ESSENTIAL CHARACTERISTICS AND ARBITRARY DETAILS - ONLY THIS TABLE AND THE FOLLOWING LINES WOULD CHANGE IF WE REPLACED THE SWITCH.

3. IN FUTURE DOCUMENTS SOME FURTHER SEGMENTATION HAS BEEN SUGGESTED.

INPUT DATA ITEM: MODE ROTARY SWITCH

ACRONYM: /MODEROT/

HARDWARE: TC-2 PANEL

DESCRIPTION: /MODEROT/ INDICATES THE SETTING OF THE MODE ROTARY SWITCH, A SIX POSITION ROTARY SWITCH ON THE TC-2 PANEL.

SWITCH NOMENCLATURE: PRES, POS, DEST, MARK, RNG/BRG, D-BHT, ALT-MSLP.

DATA TYPE: +ENUMERATION+

CHARACTERISTICS OF VALUES

VALUE ENCODING:

$None$	(000000),
$PRESPOS$	(100000),
$DEST$	(010000),
$MARK$	(001000),
RNG/BRG	(000100),
$DBHT$	(000010),
$ALTMSLP$	(000001)

INSTRUCTION SEQUENCE: READ 196 (CHANNEL 6)

DATA REPRESENTATION: TC-2 PANEL INPUT WORD 3, BIT 0-5

TIMING CHARACTERISTICS: /MODEROT/ = $None$ INDICATES THAT THE SWITCH IS IN TRANSITION BETWEEN TWO POSITIONS.

COMMENTS: THE MODE ROTARY SWITCH HAS GROWTH CAPABILITY TO EIGHT POSITIONS.

INSTRUCTOR NOTES

POINTS TO NOTE:

1. DESCRIPTION IS IN TERMS OF EXTERNALLY VISIBLE EFFECT.

2. ALL ANGULAR MEASURES ARE DESCRIBED WITH EXACTLY THE SAME TEMPLATE SO DESIGNERS AND TESTERS BECOME FAMILIAR WITH INTERPRETATION OF INFORMATION.

OUTPUT DATA ITEM: STEERING ERROR

ACRONYM: //STERROR//

HARDWARE: ATTITUDE DIRECTION INDICATOR

DESCRIPTION: //STERROR// CONTROLS THE POSITION OF THE VERTICAL NEEDLE ON THE ADI. A POSITIVE VALUE MOVES THE POINTER TO THE RIGHT WHEN LOOKING AT THE ADI. A VALUE OF ZERO CENTERS THE NEEDLE.

DATA TYPE: +ANGULAR MEASURE+

CHARACTERISTICS OF VALUES

UNIT: DEGREES

RANGE: -2.5 TO +2.5

ACCURACY: ± .1

RESOLUTION: .00122

INSTRUCTION SEQUENCE: WRITE 229 (CHANNEL 7)
TEST CARRY BIT = 0 FOR REQUEST ACKNOWLEDGED
IF NOT, RESTART

DATA REPRESENTATION: 11-BIT TWO'S COMPLEMENT NUMBER, BIT 0 AND BITS 3-12
SCALE = 512/1.25 = 409.6
OFFSET = 0

( )	NOT	USED	(				INDICATED VALUE					)			
													0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

BIT

TIMING CHARACTERISTICS: DIGITAL TO DC VOLTAGE CONVERSION. SEE SECTION 1.5.7

COMMENTS: THE POINTER HITS A MECHANICAL STOP AT ± 2.5 DEGREES.

INSTRUCTOR NOTES

ANOTHER SIMPLE, INTUITIVE CONCEPT APPLIED UNIFORMLY TO REDUCE AMBIGUITY, REDUNDANCY AND BULK.

DEFINITIONS OF TEXT MACROS CAN BE VERY COMPLEX - SOME ARE VERY MUCH LIKE MODES OR CAN TAKE ON NON-NUMERIC OR NUMERIC VALUES, MOST ARE JUST TRUE OR FALSE.

USUALLY A TEXT MACRO IS INTRODUCED AND USED BEFORE IT IS PRECISELY DEFINED. THIS IS A GOOD EXAMPLE OF STEPWISE REFINEMENT.

TEXT MACROS

- PROVIDES SINGLE, PRECISE DEFINITION OF FREQUENTLY USED, COMPLEX DESCRIPTIONS
- DEFINITION INCLUDED (ONCE) IN TEXT MACRO DICTIONARY
- ABBREVIATION, ENCLOSED IN "!" USED THROUGHOUT DICTIONARY
  - E.G., !GROUND TRACK! - LINE DEFINED BY PROJECTION OF A/C VELOCITY VECTOR INTO HORIZONTAL PLANE
- MAY INVOLVE TABLES, DIAGRAMS, EQUATIONS
- POSTPONING PRECISE DEFINITION SUPPORTS STEPWISE REFINEMENT

INSTRUCTOR NOTES

AGAIN DEMONSTRATES SEPARATION OF CONCERNS. FUNCTIONS DEFINED SEPARATE FROM INPUTS AND OUTPUTS AND SEPARATE FROM TIMING AND ACCURACY.

RELATIONSHIP OF FUNCTIONS TO OUTPUTS MUST BE STRESSED - IS A 1:N RELATION.

SUCH A SPECIFICATION HAS HUNDREDS OF SMALL FUNCTIONS, EACH CONTROLLING ONE OR FEW OUTPUTS. ONLY CONTROL MULTIPLE OUTPUTS IF THEY ALWAYS CHANGE AT THE SAME TIME.

BECAUSE OF THE RELATIONSHIP BETWEEN FUNCTIONS AND OUTPUTS IT IS ALWAYS CLEAR WHERE A PARTICULAR FUNCTIONAL REQUIREMENT WILL APPEAR.

TWO GROUPS INDEPENDENTLY DEVELOPING SPECIFICATIONS FOR THE SAME SYSTEM WILL COME UP WITH THE SAME NUMBER OF FUNCTIONS - NOT TRUE WITH CURRENT MIL-STDS. MAKES IT MUCH EASIER TO ESTIMATE SIZE AND COST OF SYSTEM.

DESCRIPTION OF SOFTWARE FUNCTIONS

- TIMING AND ACCURACY REQUIREMENTS PRESENTED SEPARATELY
- FUNCTIONS HAVE 1:N RELATIONSHIP TO OUTPUT DATA ITEMS
  - A FUNCTION MAY CONTROL 1 OR MORE OUTPUTS
  - AN OUTPUT DATA ITEM IS GIVEN VALUES BY ONLY ONE FUNCTION
  - THERE IS A FUNCTION FOR EACH INDEPENDENTLY CHANGEABLE OUTPUT OR GROUP OF OUTPUTS
- OUTPUT OF FUNCTIONS ARE IN TERMS OF ESSENTIAL CHARACTERISTICS OF OUTPUT DATA ITEMS
- INPUTS TREATED AS RESOURCES - NOT SPECIFICALLY ASSOCIATED WITH FUNCTIONS
- FUNCTIONS ARE DEMAND OR PERIODIC

INSTRUCTOR NOTES

DEMAND FUNCTIONS ONLY EXECUTE IN RESPONSE TO AN OPERATOR OR OTHER EXTERNAL ACTION. PERIODIC FUNCTIONS ARE INITIATED OR TERMINATED BY SOME EXTERNAL EVENT AND THEN PERFORM WITH A DEFINED PERIOD.

TEMPLATES FOR TWO TYPES OF FUNCTIONS ARE SOMEWHAT DIFFERENT - EXAMPLES OF BOTH ARE INCLUDED LATER.

FIRST WE WILL DISCUSS MODES WHICH SIMPLIFY THE FUNCTION DESCRIPTIONS.

DESCRIPTION OF SOFTWARE FUNCTIONS

- DEMAND FUNCTIONS - FUNCTION IS REQUESTED BY THE OCCURANCE OF AN EVENT EACH TIME IT IS PERFORMED.
  - STANDARD TEMPLATE SPECIFIES WHEN FUNCTION IS REQUESTED AND HOW OUTPUT IS PRODUCED
- PERIODIC FUNCTION - FUNCTION IS PERFORMED REPEATEDLY WITHOUT BEING REQUESTED EACH TIME
  - STANDARD TEMPLATE SPECIFIES INITIATION AND TERMINATION
  - (ADDITIONAL) STANDARD TEMPLATE SPECIFIES HOW OUTPUT IS PRODUCED
- TEMPLATES IDENTIFY CONTROLLING CONDITIONS AND EVENTS
- MODES ARE USED TO SIMPLIFY FUNCTION DESCRIPTIONS
- MODES, CONDITIONS AND EVENTS MAP DIRECTLY TO IDEF DEFINITIONS

INSTRUCTOR NOTES

NRL SPEC MANAGES TO DESCRIBE MOST FUNCTIONS IN AN END-TO-END MANNER, WITHOUT INTRODUCING INTERMEDIATE VALUES OR VARIABLES. ACTUALLY MODES AND TEXT MACROS ACT LIKE INTERMEDIATE VALUES IN MANY SITUATIONS.

FOR THE EP SPEC, SOFTECH INTRODUCED THE CONCEPT OF A SEMANTIC ENTITY WHICH IS AN OUTPUT TO A SHARED DEVICE - SUCH AS A DISPLAY. THE SEMANTIC ENTITIES ARE TREATED JUST LIKE OUTPUTS WITH ASSOCIATED FUNCTIONS CONTROLLING THEIR VALUES. OTHER FUNCTIONS CONTROL WHICH GROUPS OF SEMANTIC ENTITIES "OWN" THE SHARED DEVICE AND ARE VISIBLE AT ANY TIME.

TECHNIQUES FOR DESCRIBING SOFTWARE FUNCTIONS

- EVERY FUNCTION IS DESCRIBED IN TERMS OF EXTERNALLY VISIBLE EFFECT

- FUNCTION MAY BE ASSOCIATED WITH INDIVIDUAL OUTPUT TO SYSTEM EXTERNAL DEVICE OR TO VIRTUAL DEVICE
  - VIRTUAL DEVICES MAY BE PHYSICALLY SHARED HARDWARE DEVICE
  - VIRTUAL DEVICES ASSUME A SEPARATE FUNCTION AS NEEDED FOR EACH TYPE OF INFORMATION PRESENTED ON THE SHARED HARDWARE DEVICE

- FUNCTIONS ARE DESCRIBED IN TERMS OF CONDITIONS THAT EXIST AND EVENTS WHICH OCCUR

- EXTERNAL TO THE SYSTEM INSTEAD OF AS A FORMAL FUNCTION OF THE INPUTS
  - ELIMINATES NEED FOR A LARGE NUMBER OF INTERMEDIATE FUNCTIONS AND DATA ITEMS
  - MAKES REQUIREMENT STATEMENTS INDEPENDENT OF IMPLEMENTATION

INSTRUCTOR NOTES

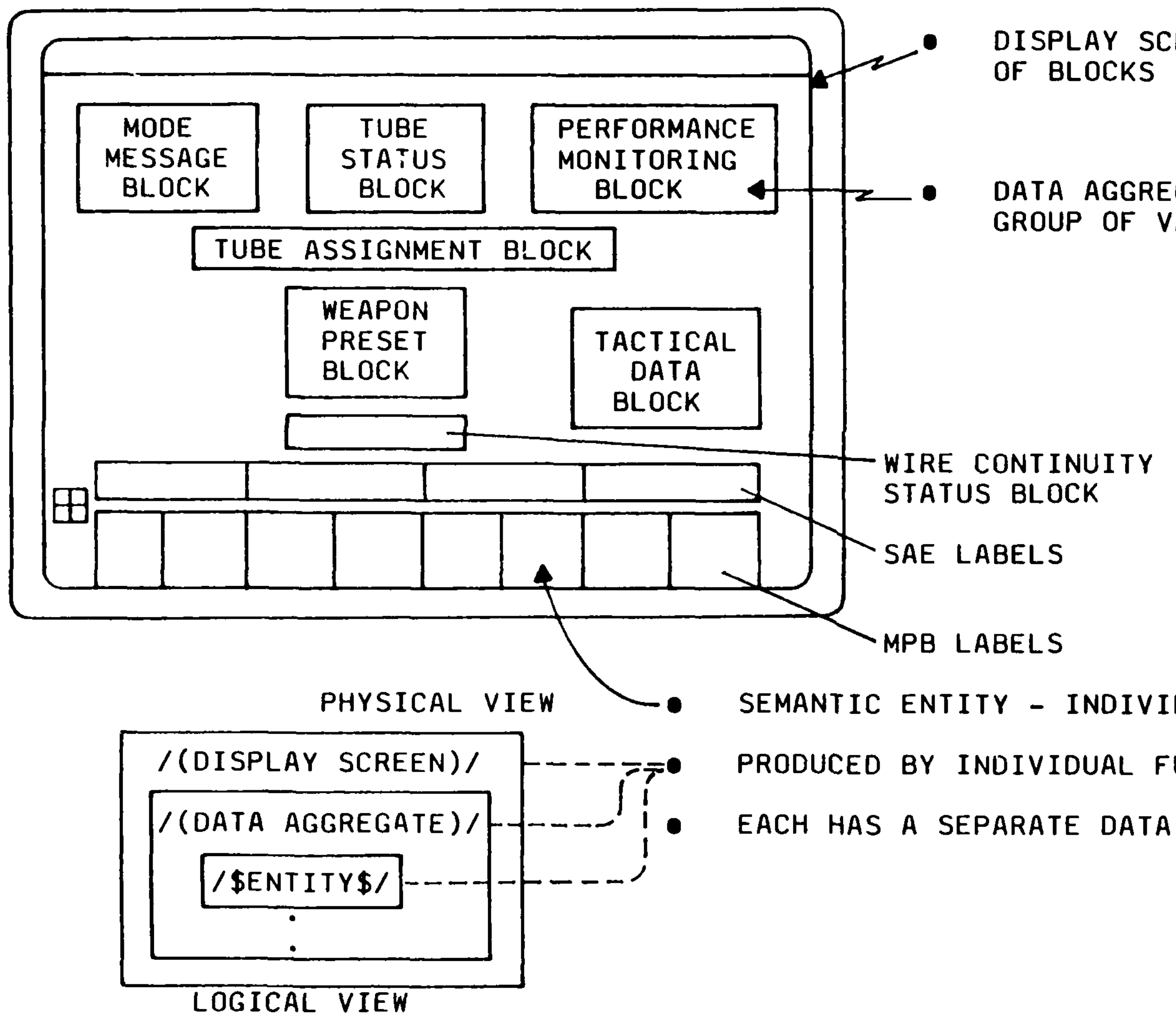
THIS EXAMPLE IS FROM EP AND SHOWS SEMANTIC ENTITIES AND GROUPS OF SEMANTIC ENTITIES CALLED A DATA AGGREGATE.

SEPARATE FUNCTIONS (USUALLY DEMAND) CONTROL WHICH DISPLAY SCREEN IS CURRENTLY VISIBLE AND THEN ANOTHER DEMAND FUNCTION CONTROLS WHICH DATA AGGREGATES ARE DISPLAYED (THIS ASSUMES THAT THE BLOCKS ON A DISPLAY SCREEN CAN COME AND GO INDEPENDENT OF THE OVERALL DISPLAY SCREEN - WHICH IS THE CASE).

YET NAME FUNCTIONS, POSSIBLY PERIODIC, ARE RESPONSIBLE FOR SETTING EACH VALUE IN A BLOCK.

VIRTUAL DEVICE EXAMPLE

- CRT DISPLAY



- DISPLAY SCREEN - A RELATED GROUP OF BLOCKS
- DATA AGGREGATE DISPLAY - A RELATED GROUP OF VALUES
- SEMANTIC ENTITY - INDIVIDUAL DISPLAYED VALUE
- PRODUCED BY INDIVIDUAL FUNCTIONS
- EACH HAS A SEPARATE DATA ITEM DESCRIPTION

VG 778.1

INSTRUCTOR NOTES

READ SECTION VI OF THE HENNINGER PAPER.

THIS APPROACH USES TRADITIONAL DEFINITIONS OF CONDITION AND EVENTS INTRODUCES @ T AND @ F TO SIGNIFY EVENTS WHICH OCCUR WHEN A CONDITION BECOMES TRUE OR FALSE.

CONDITIONS AND EVENTS

- CONDITION IS AN EXPRESSION THAT MAY BE TRUE OR FALSE
- /MODEROT/=$DEST$
- CAN USE AND, OR, NOT, WHEN
- /MODEROT/=$DEST$ OR $MARK$
- AN EVENT REPRESENTS THE MOMENT THAT A CONDITION CHANGES FROM TRUE TO FALSE OR VICE VERSA
- @T(/IMSMODE/=$Gndal$) WHEN (/ACAIRB/-YES)

INSTRUCTOR NOTES

MODES WERE INTRODUCED TO SPECS AS AN AFTERTHOUGHT TO SIMPLIFY FUNCTION DESCRIPTIONS. ORIGINAL HENNINGER PAPER CONTAINS AN OUTLINE THAT DOESN'T EVEN SHOW A MODE SECTION.

HOWEVER, MODES ARE VERY IMPORTANT TO UNDERSTANDABILITY. THEY ARE ANOTHER LEVEL OF PARTITIONING OF COMPLEXITIES AND SEPARATION OF CONCERN.

TABLES ARE USED TO PRECISELY DEFINE MODES, TO SHOW THE ALLOWABLE TRANSITIONS AND THEIR CAUSES, AND TO DEFINE THE MODES WHICH THE SYSTEM CAN START OUT IN.

MODES

- REPRESENTS A (BOOLEAN) COMBINATION OF CONDITIONS OF DATA ITEMS THAT HAVE WIDESPREAD EFFECT - (APPEAR IN MANY FUNCTION TEMPLATES)
  - CAN BE VIEWED AS REPRESENTING THE STATE OR A GROUP OF STATES OF THE SYSTEM

- MODES ARE ARBITRARY, BUT ...

- MODES ARE IMPORTANT TO UNDERSTANDABILITY (AND OPERABILITY) OF SYSTEM
  - MAY MAP TO USER NOTION OF HOW THE SYSTEM WORKS
  - SIMPLIFY FUNCTION DESCRIPTIONS BECAUSE FUNCTIONS TEND TO DIFFER MORE BETWEEN MODES THAN THEY DO IN A SINGLE MODE
  - SIMPLIFY OPERATOR INTERFACE

- MODES PRECISELY DEFINED IN MODE CONDITION TABLE

- MODE TRANSITIONS (REDUNDANTLY) SPECIFIED IN MODE TRANSITION TABLE

- INITIAL MODES DEFINED IN SEPARATE TABLE

INSTRUCTOR NOTES

WILL SHOW SIGNIFICANT ASPECTS OF EACH TYPE OF MODE TABLE AND HOW THEY ARE REDUNDANT.

EXAMPLES OF MODE TABLES

- INITIAL MODE TABLE
- MODE CONDITION TABLE
- MODE TRANSITION TABLE (PARTIAL)
- MODE TRANSITION EVENTS (PARTIAL)

INSTRUCTOR NOTES

THIS TABLE SHOWS WHAT MODE THE SYSTEM WILL COME UP IN DEPENDING ON VARIOUS SWITCH SETTINGS AND SENSOR INPUTS WHEN THE SYSTEM IS STARTED. /IMSMODE/ IS A COCKPIT SWITCH. THE VARIOUS TEXT MACROS HAVE VALUES BASED ON OTHER INPUTS.

TABLE 3.4-A: INITIAL MODES

MODES	DEFINING CONDITIONS
Landaln	!IMS Up! AND /IMSMODE/=$Gndal$ AND !Data 23! = !Land!
OLB	!IMS Up! AND /IMSMODE/=$Iner$ OR $Norm$ OR (/IMSMODE/=$Gndal$ AND !Data 23!=!Sea!)
Mag sl	!IMS Up! AND /IMSMODE/=Mag_sl
Grid	!IMS Up! AND /IMSMODE/=$Grid$
IMS fail	!IMS Down!

INSTRUCTOR NOTES

THIS TABLE SHOWS FOR EACH MODE WHAT VALUES WILL BE PRESENT FOR THE VARIOUS INPUTS AND TEXT MACROS THAT CAN CAUSE MODE CHANGES. A TABLE LIKE THIS CAN BE SCANNED TO IDENTIFY INCONSISTENT OR OVERLAPPING DEFINITIONS. GRUMMAN HAS INTRODUCED A SLIGHTLY DIFFERENT AND EXPANDED TABLE WHICH HAS A 1 OR 0 IN EACH TABLE ENTRY ALLOWING EASIER CROSS CHECKING.

TABLE 3.3-a: NAVIGATION MODE CONDITIONS

Condition / Mode	/IMSMODE/=	/ACAIRB/=	Align. stage completed	!latitude!	Other
DIG	$Norm$ OR $Gndal$	Yes	!CA stage!	ls 70o	!Doppler up! AND !IMS up!
DI	$Iner$ OR $Norm$ OR $Gndal$	Yes	!CL stage!	ls 80o	!Doppler used! AND !IMS up!
I	$Iner$ OR $Norm$ OR $Gndal$	X	!CL stage!	ls 80o	NOT !Doppler used! AND !IMS up!
Mag sl	$Magsl$	X	X	ls 80o	!IMS up!
Grid	$Grid$	X	X	X	!IMS up!
UDI	$Iner$	Yes	NOT !CL stage!	ls 80o	!IMS up! AND !Doppler used! AND !pitch small! AND !roll small!
OLB	$Iner$ OR $Norm$ OR $Gndal$ OR	X	NOT !CL stage!	ls 80o	X
IMS fail	X	X	X	X	!IMS down!
PolarDI	$Iner$ OR $Norm$ OR $Gndal$	Yes	!CL stage!	X	!Doppler used! AND !IMS up!
PolarI	$Iner$ OR $Norm$ OR $Gndal$	X	!CL stage!	X	NOT !Doppler used! AND !IMS up!

INSTRUCTOR NOTES

THIS TABLE SHOWS THE POSSIBLE MODE TRANSITIONS - THE ACTUAL CAUSE IS SHOWN IN THE NEXT SLIDE. EACH ROW/COLUMN INTERSECTION WITH A NUMBER SHOWN IS A POSSIBLE MODE TRANSITION. THE NUMBER IS THE ROW AND COLUMN (ALSO INDICATES THE EXITING MODE # AND THE ENTERING MODE #) AND IS USED AS AN IDENTIFIER IN THE FOLLOWING SLIDE TO IDENTIFY THE CAUSING EVENT.

TRANSITION TABLE 3.4: TRANSITIONS BETWEEN ALIGNMENT, CALIBRATION AND NAVIGATION MODES

EXITED MODE

Lautocal			1-3			
Sautocal		2-2		2-4		
Landaln	3-1		3-3	3-4		3-6
SINSaln		4-2	4-3	4-4		
OlUpdate	5-1		5-3			5-6
HUDaln	6-1					6-6
Airaln						
DIG						
DI						
I			10-3	10-4	10-5	10-6
UDI						
OLB			12-3	12-4		12-6
Mag sl			13-3	13-4		
Grid			14-3	14-4		
IMS fail			15-3			
PolarDI						
PolarI			17-3	17-4	17-5	
Grtest			18-3	18-4		
ENTERED MODE	*Lauto cal*	*Sauto cal*	*Landaln*	*SINSaln*	*Ol Update*	*HUDaln*

INSTRUCTOR NOTES

THIS MODE TRANSITION EVENT SECTION OR TABLE SHOWS THE EXACT CAUSE (OR POTENTIAL CAUSES) FOR EACH MODE TRANSITION IDENTIFIED IN THE PREVIOUS SLIDE.

```
1-3   *Lautocal*  TO *Landaln*
          @T(!present position entered!)
1-7   *Lautocal* TO *Airaln*
          @T(/ACAIRB/=$Yes$) WHEN (!Doppler up!)
1-10  *Lautocal*  TO  *I*
          @T(!ND stage! completed) WHEN (!latitude! lseq 80o)
1-12  *Lautocal*  TO *OLB*
          @T(!present position entered!) WHEN (/IMSMODE/-$Norm$ OR $Iner$)
          @T(/ACAIRB/=$Yes$)  WHEN (!Doppler down!)
          @T(!present position entered!)
1-13  *Lautocal* TO *Mag sl*
          @T(!present position entered! WHEN(/IMSMODE/=$Magsl$)
1-14  *Lautocal* TO *Grid*
          @T(!present position entered!) WHEN(/IMSMODE/=$Grid$)
1-15  *Lautocal* TO *IMS fail*
          @T(!IMS down!)
1-17  *Lautocal*  TO  *PolarI*
          @T(!ND stage! completed) WHEN (!latitude! gt 80o)
1-18  *Lautocal* TO *Grtest*
          @T(/PNLTEST/=$TEST$)
2-2   *Sautocal*  TO *Sautocal*
          @T(NOT !SINS valid! for more than 2 sec)
2-4   *Sautocal*  TO *SINSaln*
          @T(!present position entered!)
2-7   *Sautocal* TO *Airaln*
          @T(/ACAIRB/=$Yes$)  WHEN (!Doppler up!)
2-10  *Sautocal*  TO  *I*
          @T(!ND stage!   completed) WHEN (!latitude! lseq 80o)
2-12  *Sautocal*  TO *OLB*
          @T(/ACAIRB/=$Yes$)  WHEN (!Doppler down!)
          @T(!present position entered!)WHEN (/IMSMODE/=$Norm$ OR $Iner$)
```

INSTRUCTOR NOTES

THE FOLLOWING SLIDES WILL INTRODUCE THE TEMPLATES USED TO DESCRIBE DEMAND AND PERIODIC FUNCTIONS.

SOFTWARE FUNCTION EXAMPLES

- DEMAND FUNCTIONS TEMPLATE
- PERIODIC FUNCTION TEMPLATE
  - INITIATION AND TERMINATION
  - OUTPUT DESCRIPTION

INSTRUCTOR NOTES

ALL FUNCTION DESCRIPTIONS INDICATE THE ASSOCIATED OUTPUT DATA ITEMS PRODUCED BY THE FUNCTION, THUS PROVIDING A CROSS REFERENCE TO THE DATA ITEM DESCRIPTIONS. THE LIST OF MODES IS TO PROVIDE THE READER WITH AN OVERVIEW OF WHEN THE FUNCTION IS PERFORMED, THIS OVERVIEW IS REFINED IN THE FUNCTION DESCRIPTION PORTION OF THE TEMPLATE.

THE EVENT TABLE SHOWS THE EVENTS (I.E., A TRANSITION OF A CONDITION) THAT REQUEST THE FUNCTION TO BE PERFORMED AND THE MANNER IN WHICH THE OUTPUT IS TO BE PRODUCED. SYMBOLIC NAMES ARE USED FREELY TO PROVIDE AN INDEPENDENCE BETWEEN VALUES AND NAMES.

4.1.5 Demand Function Name: Change scale factor

Modes in which function required:

Alignment: *Lautocal*, *Sautocal*, *LandAln*, *SINSAln*, *OlUpdate*, *HUDAln*

Navigation: *DIG*, *DI*, *I*, *UDI*, *OLB*, *PolarDI*, *PolarI*

Output data item: //IMSSCAL//

Function Requested and Output Description:

EVENT TABLE 4.1-h: WHEN THE SCALE FACTOR IS CHANGED

MODES	EVENTS	
Lautocal *Landaln* *Ol Update*	@T(In mode) WHEN (//IMSSCAL// =$Coarse$)	X
HUDaln	@T(In mode) WHEN (/IMSMODE/ = $Gndal$ AND //IMSSCAL//=$Coarse$)	@T(In mode) WHEN (/IMSMODE/ = ($Norm$ OR $Iner$) AND //IMSSCAL// = $Fine$)
Sautocal *SINSaln* *Airaln* All navigation modes listed	X	@T(In mode) WHEN (//IMSSCAL//=$Fines$)
ACTION	//IMSSCAL//:=$Fine$	//IMSSCAL//:=$Coarse$

INSTRUCTOR NOTES

THE STRUCTURE OF THE PERIODIC FUNCTION TEMPLATE IS SIMILAR TO THE DEMAND FUNCTION TEMPLATE.

THE EVENT TABLE HERE IDENTIFIES THE EVENTS THAT INITIATE AND TERMINATE PERIODIC PROCESSING OF THIS FUNCTION. THE RATE AT WHICH IT IS PERFORMED IS SPECIFIED IN THE TIMING SECTION OF THE SPECIFICATION (I.E., SEPARATION OF TIMING CONCERNS FROM FUNCTIONAL CONCERNS).

PERIODIC FUNCTIONS ONCE INITIATED ARE ASSUMED TO BE PERFORMED AT THE RATE SPECIFIED AND REQUIRE NO FURTHER EVENT TO FORCE THEIR COMPUTATION.

OUTPUT DESCRIPTION DESCRIBES HOW THE OUTPUT DATA ITEM IS COMPUTED.

4.3.8 Periodic function name: Update Pullup Anticipation Cue Coordinates

Modes in which function required:

Weapon deliver:	*Nattack*, *Noffset*, *BOC*, *BOCFlytoO*, *BOCoffset*, *CCIP*, *A/G Guns*, *Shrike*, *Walleye*, *Snattack*, *Snoffset*, *SBOC*, SBOCflytoO*, *SBOCoffset*
Test:	*Grtest*

Initiation and Termination Events:
In the following table, !Range! refers to ground range to the !FLY-TO-point!

Event Table 4.3.8-a: When PUAC Updated (A)

MODES	EVENTS	
Nattack, *Noffset*, *BOCGlytoO*, *CCIP*, *A/G Guns*	@T (In mode)	X
*BOC, *BOCoffset*	@T(In mode AND !Range! lseq 30 nmi)	@T(!Range! gt 30 nmi)
Grtest	@F(!No WD MFS!)	@T(!No WD MFS!)
None of the listed modes	X	@T(!No WD MFS!)
ACTION	Initiation PUAC in view	Termination PUAC out of view

Output Data Items: //PUACAZ//, //PUACEL//

Output description:

In most modes, the PUAC shows the pilot how far he is from the "pullup point": the point where he must execute a 4g pullup to avoid either the ground or the blast radius of a released weapon.

INSTRUCTOR NOTES

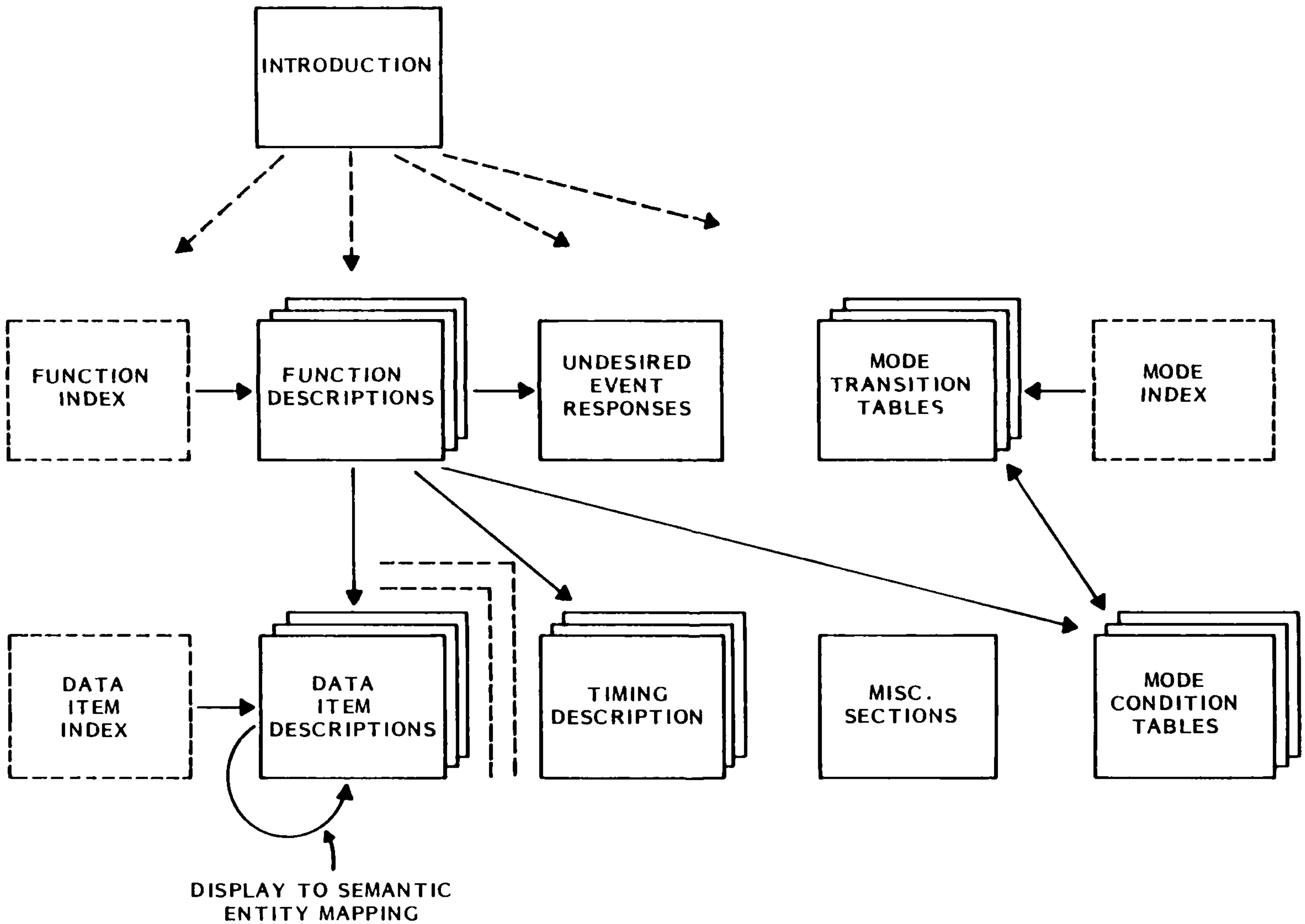
THIS SHOWS THE STRUCTURE OF A SCRP SPECIFICATION AND THE RELATIONSHIPS BETWEEN SECTIONS. SECTIONS THAT SHOULD BE NOTED ARE:

- INTRODUCTION - PROVIDES A READER'S GUIDE TO THE DOCUMENT AND THE SYSTEM. SOME PEOPLE HAVE INCLUDED A GRAPHICAL MODEL (LIKE SADT) TO SUPPLEMENT THE INTRODUCTION.

- UNDESIRED EVENT RESPONSES - LISTS THE EFFECTS OF VARIOUS ABNORMAL OR FAILURE TYPE EVENTS ON THE SYSTEM (I.E., POWER FAILURE, STACK OVERFLOW, ETC.). AGAIN SEPARATES THE NORMAL FUNCTIONAL CONCERNS FROM THE ABNORMAL ONES.

THE ACTUAL PACKAGING OF THE SECTIONS MAY BE DICTATED BY MIL-STD REQUIREMENTS AND ARE NOT COVERED HERE.

SCRP SPECIFICATION PACKAGING



INSTRUCTOR NOTES

THIS SLIDE DETAILS THE CONTENT OF THE VARIOUS SECTIONS. POINT OUT THAT <u>ALL</u> SPECIFICATIONS MUST HAVE THIS INFORMATION IN THEM, ALTHOUGH MAY NOT BE ORGANIZED IN THIS MANNER.

SUMMARY OF REQUIREMENTS SPECIFICATION PACKAGING

SECTION	INCLUDES
• INTRODUCTION	- GUIDE TO READING THE DOCUMENT - REFERENCE (POSSIBLE INCLUSION) OF IDEF MODELS TO GIVE OVERVIEW DESCRIPTION OF SYSTEM
• DISTINGUISHING CHARACTERISTICS OF THE COMPUTER ENVIRONMENT	- GENERAL DESCRIPTION (IF SELECTED) - SUMMARY OF REQUIRED CHARACTERISTICS (IF NOT SELECTED) - IDENTIFICATION OF IDIOSYNCRASIES
• INPUT AND OUTPUT DATA ITEMS	- CORRESPONDS TO A DATA DICTIONARY OF THE SYSTEM INPUTS AND OUTPUTS - DATA TYPE DEFINITIONS IN TABULAR FORMAT - SET OF DATA ITEM DESCRIPTIONS FOR SYSTEM I/O - INTERMEDIATE DATA ITEMS USING SIMILAR DATA ITEM DESCRIPTIONS (CLEARLY MARKED AS INTERMEDIATES)
• MODES OF OPERATION	- A CONCISE DEFINITION OF THE MAJOR OPERATIONAL MODES OF THE SYSTEM
• TIME INDEPENDENT DESCRIPTIONS OF FUNCTIONS	- FUNCTIONAL REQUIREMENTS OFTEN CALLED "PERFORMANCE REQUIREMENTS" - SET OF FUNCTION TEMPLATES

INSTRUCTOR NOTES

NOTE EXPECTED TYPES OF CHANGES SECTION SINCE THIS WILL GIVE GUIDANCE TO THE SOFTWARE DESIGNERS AS TO HOW TO MODULARIZE THE SOFTWARE THAT IMPLEMENTS THE REQUIREMENTS. POINT OUT THAT THE GOAL IS TO HAVE EXPECTED CHANGES HIDDEN IN A SINGLE MODULE IN THE DESIGN TO MINIMIZE MAINTENANCE LIFE-CYCLE COSTS.

SUMMARY OF REQUIREMENTS
SPECIFICATION PACKAGING - continued

SECTION	INCLUDES
• TIMING REQUIREMENTS	- STATE HOW OFTEN AND HOW FAST FUNCTIONS MUST BE PERFORMED - OPTIONALLY MAY SPECIFY DEGREE OF CONCURRENCY, DATA TRANSFER RATES, ETC.
• ACCURACY CONSTRAINTS ON FUNCTIONS	- IDENTIFIES HOW CLOSE OUTPUT VALUES MUST BE TO IDEAL VALUES TO BE ACCEPTABLE
• UNDESIRED EVENT RESPONSES	- RESPONSES TO UNDESIRED EVENTS SUCH AS RESOURCE FAILURES (TEMPORARY AND PERMANENT), INCORRECT INPUT DATA, INCORRECT INTERNAL DATA, SOFTWARE ERRORS, POWER FAILURE, ETC.
• REQUIRED SUBSETS	- DEFINE THE MINIMAL SUBSETS OF THE SYSTEM THAT WOULD BE USEFUL OPERATIONALLY AND DURING DIFFERENT PHASES OF INTEGRATION AND TESTING
• EXPECTED TYPES OF CHANGES	- IDENTIFY THE TYPES OF CHANGES EXPECTED IN THE SOFTWARE OVER IT'S LIFETIME - IDENTIFY FUNDAMENTAL ASSUMPTIONS ABOUT THE SOFTWARE WHICH ARE VERY UNLIKELY TO CHANGE

INSTRUCTOR NOTES

SUMMARY OF REQUIREMENTS
SPECIFICATION PACKAGING - continued

- GLOSSARY, ETC. - SHOULD INCLUDE -
  - GLOSSARY OF ABBREVIATIONS, ACRONYMS, TECHNICAL TERMS
  - TEXT MACRO DICTIONARY
  - CROSS REFERENCES BETWEEN IDEF MODELS AND THIS SPECIFICATION, MODES AND FUNCTIONS, DATA ITEMS AND FUNCTIONS, TEXT MACROS AND FUNCTIONS, EVENTS, CONDITIONS AND MODES
- SOURCES FOR FURTHER INFORMATION - ANNOTATED LIST OF DOCUMENTATION AND PERSONNEL INDICATING THE TYPES OF QUESTION EACH ANSWERS

INSTRUCTOR NOTES

IN THIS WRAP-UP MAKE IT CLEAR THAT THE SCRP METHODOLOGY IS <u>REAL</u> AND BEING USED. ALSO THAT IT HAS SUPPORTING METHODS FOR DESIGN THAT WE WILL SEE IN THE DESIGN SECTION OF THE COURSE.

REQUIREMENTS SPECIFICATION SUMMARY

- DOCUMENT IS PRECISE AND CONCISE
- COMPLETENESS CAN BE VERIFIED
- TEMPLATES, STRUCTURE SUPPORTS REVIEW
- COMPLEXITY PARTITIONED - SEPARATION OF CONCERNS, MODULARITY
- ALLOWS MANY PARTICIPANTS AT VARIED EXPERIENCE LEVELS

INSTRUCTOR NOTES

<u>THEME</u>: THE ANALYSIS TECHNIQUES PRESENTED ADDRESS DIFFERENT PORTIONS OF THE ANALYSIS PHASE AS WELL AS THE USABILITY OF THE METHODS FOR DIFFERENT SITUATIONS.

<u>PURPOSE</u>: TO PROVIDE A FRAMEWORK FOR THE STUDENTS TO AID IN THE SELECTION OF ANALYSIS METHODS.

<u>REFERENCES</u>: Ada METHODOLOGIES: CONCEPTS AND REQUIREMENTS/METHODMAN DOCUMENTS, DEPARTMENT OF DEFENSE, AJPO, 1983.

Section 12
ANALYSIS WRAP-UP

VG 778.1

INSTRUCTOR NOTES

- COMMUNICATE → 1st YOURSELF, 2nd OTHERS, 3rd COMPROMISE
- DESIGN → 1st UNDERSTAND, 2nd SOLUTIONS, 3rd HOWS
- EXPLICIT → WRITE EVERYTHING DOWN

METHODS → GRAPHICS, RIGOR, *APPROPRIATE* LANGUAGE

- QUALITY → CUSTOMER AND PEER REVIEWS

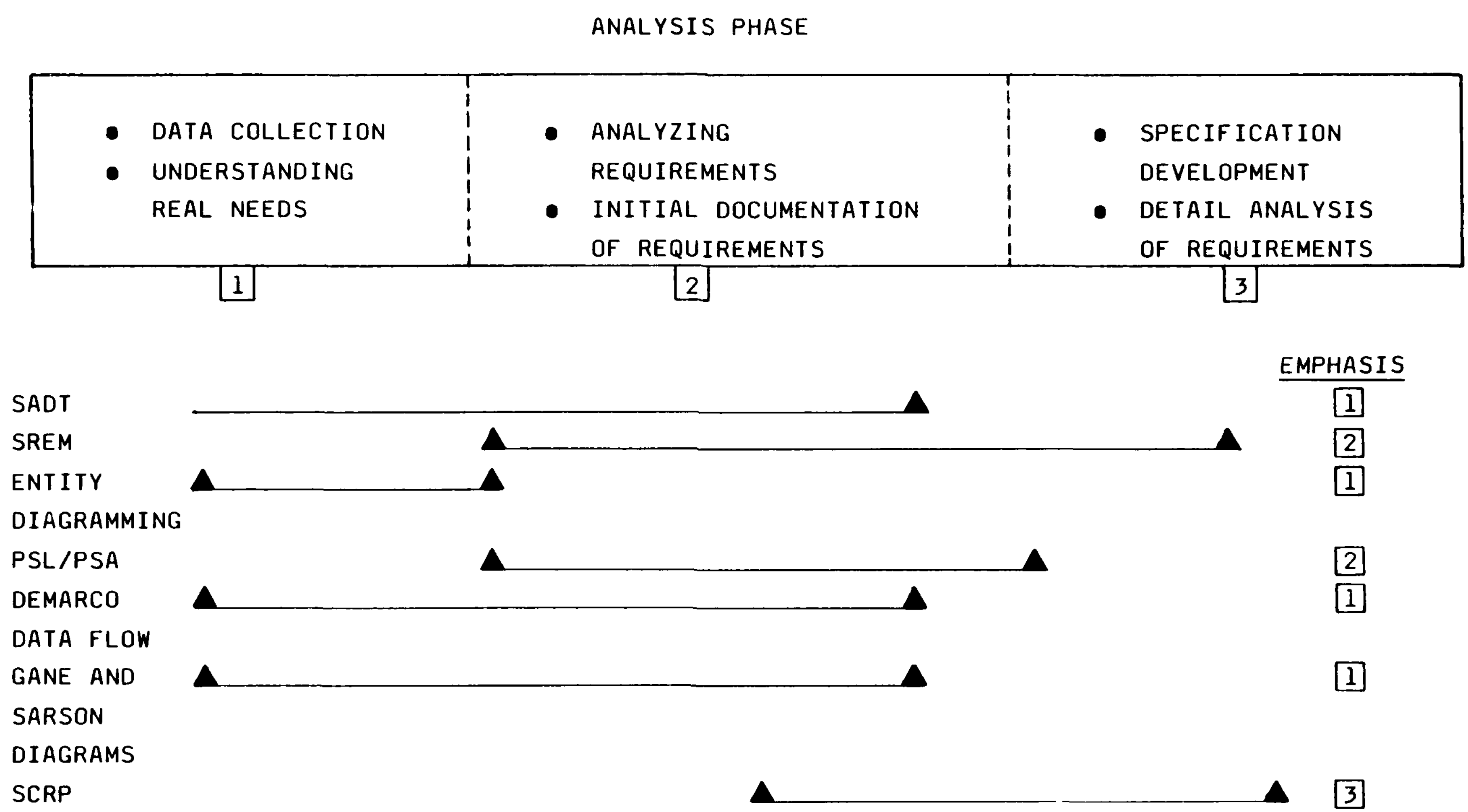
POINTS

- DURING ANALYSIS ...
  - COMMUNICATE
  - DON'T DESIGN
  - BE EXPLICIT
  - USE EFFECTIVE METHODS
  - STRIVE FOR QUALITY

INSTRUCTOR NOTES

SUGGEST USING CERTAIN TECHNIQUES TOGETHER, SUCH AS SADT AND SCRP, GIVES MAXIMUM COVERAGE

ANALYSIS PHASE COVERAGE



INSTRUCTOR NOTES

THE SET OF CRITERIA IS FROM THE METHODMAN DOCUMENT WHICH DESCRIBED CHARACTERISTICS OF METHODOLOGIES THAT SHOULD BE IN SOFTWARE ENGINEERING ENVIRONMENTS.

METHODOLOGY EVALUATION CRITERIA

- THIS SET OF CRITERIA ALLOWS US TO LOOK AT THE METHODOLOGY FROM THEIR ABILITY TO SUPPORT OUR NEEDS
- THREE CATEGORIES OF CRITERIA
  - TECHNICAL CHARACTERISTICS
  - USAGE CHARACTERISTICS
  - MANAGEMENT CHARACTERISTICS
- A QUALITATIVE ASSESSMENT OF EACH METHODOLOGY IS PROVIDED HERE
- CRITERIA WILL BE COMMON FOR ALL METHODOLOGIES IN THIS COURSE

CRITERIA SOURCE: METHODMAN, DoD, AJPO, 1983.

INSTRUCTOR NOTES

THE BOXES HIGHLIGHT THE CHARACTERISTICS NEEDED FOR ANALYSIS

TECHNICAL CHARACTERISTICS
(ANALYSIS)

- FUNCTION HIERARCHY - A SITUATION CAN BE REPRESENTED IN WHICH A FUNCTION AT ONE LEVEL IS ACTUALLY COMPOSED OF SEVERAL INTERCONNECTED FUNCTIONS THAT EXIST AT A LEVEL OF GREATER DETAIL.
- DATA HIERARCHY - THE CONCEPT OF DATA CLASSES; A SITUATION CAN BE REPRESENTED IN WHICH A SET OF DATA AT ONE LEVEL IS ACTUALLY COMPOSED OF SEVERAL INTERRELATED PIECES OF DATA THAT EXIST AT A LEVEL OF GREATER DETAIL.
- INTERFACES - THE CONCEPT OF HAVING DISTINCT AND WELL-DEFINED BOUNDARIES BETWEEN PROCESSES OR SETS OF DATA. SOFTWARE SYSTEMS AND THE LARGE INFORMATION SYSTEMS OF WHICH THEY ARE A PART SHOULD CONTAIN MANY DISTINCT PARTS. EACH PART SHOULD HAVE A CLEAR DEFINITION SO THAT IT CAN BE DEALT WITH AS A SEPARABLE UNIT. IF THIS IS THE CASE, THEN WE HAVE INTERFACES, OR CONNECTIONS, BETWEEN THE VARIOUS PARTS.
- CONTROL FLOW - REPRESENTATION OF THE SEQUENCE IN WHICH PROCESSES WILL TAKE PLACE.
- DATA FLOW - REPRESENTATION OF THE FLOW OF INFORMATION TYPES BETWEEN VARIOUS PROCESSING ELEMENTS AND/OR STORAGE ELEMENTS IN THE SYSTEM.
- DATA ABSTRACTION - THE CONCEPT OF HIDING INFORMATION ABOUT THE IMPLEMENTATION OF A DATA TYPE AND PROVIDING A SET OF IMPLEMENTATION-INDEPENDENT FUNCTIONS FOR USE OF THE DATA TYPE.
- PROCEDURAL ABSTRACTION - AN ALGORITHM FOR CARRYING OUT SOME OPERATION IS ABSTRACTED TO A SINGLE NAME THAT CAN BE USED TO INVOKE A PROCEDURE WITHOUT KNOWING THE DETAILS OF ITS IMPLEMENTATION. (TRANSACTIONS ARE AN INSTANCE OF THIS CASE.)
- PARALLELISM - A SITUATION IN WHICH TWO OR MORE COOPERATING SEQUENTIAL PROCESSES ARE CONCURRENTLY IN EXECUTION.
- SAFETY - THE AVOIDANCE OF RUNTIME FAILURES WHICH COULD LEAD TO THE LOSS OF LIFE OR THE OCCURRENCE OF OTHER CATASTROPHIC CONSEQUENCES.
- RELIABILITY - THE ABSENCE OF ERRORS THAT LEAD TO SYSTEM FAILURE.
- CORRECTNESS - FIDELITY TO HIGHER LEVEL SPECIFICATIONS.

INSTRUCTOR NOTES

WALK THROUGH THE TABLE A COLUMN AT A TIME

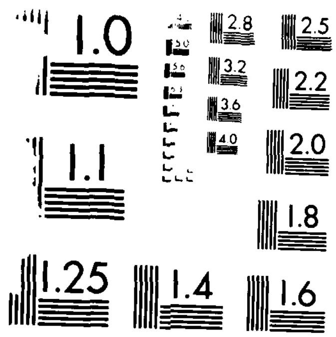
ASK THE CLASS IF THEY AGREE WITH THE LEVELS OF SUPPORT

EMPHASIS IS ON COMPARISON OF METHODOLOGIES

- *HIGHLIGHT THE METHODOLOGIES WITH THE MOST H'S*

AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE 2/6
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

UNCLASSIFIED F/G 5/9 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

TECHNICAL CHARACTERISTICS

	FUNCTION HIERARCHY	DATA HIERARCHY	INTERFACES	CONTROL FLOW	DATA FLOW	PARALLELISM	CORRECTNESS
SADT	H	H	H	M	M	M	M
SREM	H	H	H	H	H	H	H
ENTITY DIAGRAMMING		H	M				L
PSL/PSA	M	M	H	L	M	M	H
DEMARCO DATA FLOW	L	H	H	L	H	M	L
GANE AND SARSON DIAGRAMS	M	M	H	L	H	M	L
SCRP	M	M	H	H	H	H	M

LEVEL OF SUPPORT
H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

THE BOXES HIGHLIGHT THE METHODS USAGE CHARACTERISTICS NEEDED IN THE ANALYSIS PHASE.

USAGE CHARACTERISTICS
(ANALYSIS)

- UNDERSTANDABILITY - HOW EASY IT IS FOR SOMEONE WHO IS INTERESTED IN THE SYSTEM BEING DEVELOPED, BUT NOT ESPECIALLY KNOWLEDGEABLE IN THE TECHNIQUE, TO UNDERSTAND THE RESULTS OF THE DEVELOPMENT.

- TRANSFERABILITY - THE DEGREE TO WHICH THE METHOD OR TOOL CAN BE SUCCESSFULLY TAUGHT TO PERSONNEL NOT PREVIOUSLY FAMILIAR WITH IT. THIS INCLUDES NOT ONLY HOW EASY IT IS TO TEACH AND LEARN, BUT ALSO HOW WELL FORMULATED IT IS.

- REUSABILITY - THE EASE WITH WHICH PREVIOUSLY CREATED DESIGNS, CODE, OR OTHER WORK PRODUCTS CAN BE REUSED IN A NEW PROJECT.

- COMPUTER SUPPORT - EXISTENCE OF AUTOMATED TOOLS WHICH CAN BE EASILY OBTAINED AND WHICH AID IN THE USE OF THE METHODOLOGY OR SOME OF ITS STEPS.

- LIFE-CYCLE RANGE - THE SPAN OF PHASES IN THE DEVELOPMENT LIFE-CYCLE OVER WHICH THE TOOLS OR METHOD CAN BE USEFULLY APPLIED. THIS MUST BY NECESSITY BE AN APPROXIMATE MEASURE.

- TASK RANGE - THE SPAN OF TASKS TO WHICH THE ITEM MAY BE USEFULLY APPLIED. TRADITIONAL CLASSIFICATIONS OF SOFTWARE APPLICATIONS (BUSINESS, SCIENTIFIC, SYSTEMS, ETC.) ARE NOT VERY USEFUL FOR THIS EVALUATION. THERE ARE SCIENTIFIC AND BUSINESS PROBLEMS THAT SHARE THE SAME PROBLEMS WHILE THERE ARE OTHER TASKS THAT SAY, WITHIN THE BUSINESS CATEGORY, ARE QUITE DISSIMILAR.

- COHESIVENESS - THE EXTENT TO WHICH THE TECHNICAL METHODS, MANAGEMENT PROCEDURES, AND AUTOMATED TOOLS MAY BE COMBINED TO SUPPORT THE METHODOLOGY.

- EXTENT OF USAGE - A JUDGEMENT AS TO HOW WIDESPREAD THE CURRENT USAGE OF THE METHODOLOGY IS.

INSTRUCTOR NOTES

USAGE CHARACTERISTICS (ANALYSIS) - continued

- EASE OF PHASE TRANSITION - THE EXTENT TO WHICH INFORMATION DEVELOPED AT ONE PHASE OF DEVELOPMENT SUPPORTS WORK TO BE DONE AT A SUBSEQUENT PHASE (EG.., FROM ANALYSIS TO DESIGN).

- DECISION HIGHLIGHTING - THE ABILITY OF A TECHNIQUE TO MAKE VISIBLE AND HIGHLIGHT KEY TECHNICAL OR PROJECT DECISIONS (E.G., CHOICE OF DATA STRUCTURES, NEARNESS TO COMPLETION OF A PROJECT); THE ABILITY OF A TECHNIQUE TO ILLUMINATE THE CONSEQUENCES OF A DEVELOPMENT DECISION.

- VALIDATION - THE EXTENT TO WHICH THE METHODOLOGY ASSISTS IN THE DETERMINATION OF SYSTEM CORRECTNESS.

- REPEATABILITY - THE EXTENT TO WHICH SIMILAR RESULTS ARE OBTAINED WHEN THE METHODOLOGY (OR AN INCLUDED ASPECT) IS APPLIED MORE THAN ONCE TO THE SAME PROBLEM.

- EASE OF CHANGE TO WORK PROCEDURES - THE AMOUNT OF EFFORT REQUIRED TO MODIFY A WORK PRODUCT WHEN SOME ASPECT OF REQUIREMENTS, SPECIFICATION, OR DESIGN IS CHANGED.

INSTRUCTOR NOTES

WALK THROUGH A COLUMN AT A TIME

ASK CLASS IF THEY AGREE OR DISAGREE WITH RATINGS

USAGE CHARACTERISTICS

	TASK RANGE								
	EMBEDDED SYSTEMS	SCIENTIFIC/ ENGINEERING	OPERATING SYSTEMS	TOOLS	BUSINESS	ARTIFICIAL INTELLIGENCE	SMALL SIZE	MEDIUM SIZE	LARGE SIZE
SADT	W	W	W	W	S	W	S	W	W
SREM	W	W	W	S	S	W	I	S	W
ENTITY DIAGRAMMING	S	I	S	I	W	W	W	S	I
PSL/PSA	W	S	S	I	S	I	I	S	W
DEMARCO DATA FLOW	S	S	I	S	W	I	W	S	I
GANE AND SARSON DIAGRAMS	S	S	I	S	W	I	W	S	I
SCRP	W	S	W	S	S	W	W	W	W

LEGEND
W - WELL SUITED
S - SATISFACTORY
I - INAPPROPRIATE

INSTRUCTOR NOTES

MAKE THE POINT THAT UNDERSTANDABILITY AND TRANSFERABILITY ARE IMPORTANT AND FOR ANALYSIS METHODS MAY BE MORE IMPORTANT THAN TECHNICAL CHARACTERISTICS

USAGE CHARACTERISTICS

	UNDERSTANDABILITY	TRANSFERABILITY	COMPUTER SUPPORT	EXTENT OF USAGE	EASE OF PHASE TRANSITION	VALIDATION	REPEATABILITY
SADT	H	H	H	H	M	L	M
SREM	L	L	H	M	M	H	M
ENTITY DIAGRAMMING	H	H	M	H	M	L	M
PSL/PSA	L	L	H	H	L	H	H
DEMARCO DATA FLOW	H	H	M	M	M	L	M
GANE AND SARSON DIAGRAMS	H	M	L	L	M	L	M
SCRP	H	M	M	M	H	H	H

LEVEL OF SUPPORT
H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

THE BOXES HIGHLIGHT THE MANAGEMENT CHARACTERISTICS A METHOD SHOULD POSSESS IN ANALYSIS

MANAGEMENT CHARACTERISTICS

- MANAGEABILITY - THE DEGREE TO WHICH THE METHOD OR TOOL PERMITS STANDARD MANAGEMENT TECHNIQUES OF ESTIMATION, IN-PROCESS STATUS CHECKING AND CONTROL TO BE APPLIED. THE EVALUATIONS ARE BASED ON THE EXISTENCE OF WELL-DEFINED STEPS AND INTERMEDIATE PRODUCTS WHICH MAKE MANAGEMENT POSSIBLE AND THE EXISTENCE OF MANAGEMENT TECHNIQUES APPLYING SPECIFICALLY TO THE METHOD AT HAND.
- TEAMWORK - THE EXTENT TO WHICH THE METHODOLOGY AND THE DEVELOPMENT ENVIRONMENT AID, RATHER THAN HINDER, TEAMWORK.
- PHASE DEFINITIONS - THE IDENTIFICATION WITHIN THE METHODOLOGY OF DEVELOPMENT PHASES THAT REPRESENT INTERMEDIATE STAGES OF THE DEVELOPMENT PROCESS.
- WORK PRODUCTS - THE DOCUMENTS AND SYSTEMS THAT RESULT FROM APPLICATION OF THE METHODOLOGY.
- CONFIGURATION MANAGEMENT - THE WAY IN WHICH THE METHODOLOGY AND/OR ITS TOOLS PROVIDES FOR ORGANIZATION, TRACKING, AND MAINTENANCE OF THE EMERGING WORK PRODUCTS, INCLUDING CONTROL OF RELEASES AND MULTIPLE VERSIONS.
- EXIT CRITERIA - THE WAY IN WHICH PHASES OF DEVELOPMENT AND WORK PRODUCTS ARE DEFINED TO PROVIDE EXPLICIT STOPPING OR EXIT CRITERIA FOR EACH STAGE OF DEVELOPMENT.
- SCHEDULING - THE METHODOLOGICAL SUPPORT FOR PROJECT SCHEDULING.
- COST ESTIMATION - THE METHODOLOGICAL SUPPORT FOR COST ESTIMATION.

INSTRUCTOR NOTES

WALK THROUGH COLUMNS

ASK THE CLASS TO COMMENT ON THE RATINGS

MANAGEMENT CHARACTERISTICS

	MANAGEABILITY	TEAMWORK	PHASE DEFINITIONS	CONFIGURATION MANAGEMENT	EXIT CRITERIA
SADT	H	H	M	M	M
SREM	H	M	H	M	M
ENTITY DIAGRAMMING	M		L		
PSL/PSA	L	L	M	M	L
DEMARCO DATA FLOW	L	M	L	L	L
GANE AND SARSON DIAGRAMS	L	M	L	L	L
SCRP	M	M	M	M	H

<u>LEVEL OF SUPPORT</u>
H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

EMPHASIZE THAT ANALYSIS IS FAR AWAY FOR CODING, SO RELATIONSHIP MAY BE LOOSE AT THIS POINT.

ENTERTAIN QUESTIONS, BUT DON'T GET STUCK ON DETAILS. KEEP TALK ON A MORE GENERAL LEVEL RATHER THAN SPECIFIC.

ANALYSIS METHODOLOGIES AND Ada RELATIONSHIP

- ANALYSIS IS ORIENTED TOWARD SPECIFYING THE REQUIREMENTS OF A SYSTEM THEREFORE
  - CONTAINS VERY LITTLE IMPLEMENTATION DETAIL
  - HAS VERY LITTLE DIRECT CORRESPONDENCE TO Ada

- MUST PROCEED FROM ANALYSIS TO DESIGN AND IMPLEMENTATION TO ESTABLISH CLEAR RELATIONSHIPS

- POSSIBLE Ada RELATIONSHIPS
  - ABILITY TO IDENTIFY "USER DEFINED" DATA TYPE
  - MAP BETWEEN USER FUNCTIONS AND Ada SUBPROGRAMS
  - PROVIDES GUIDANCE INTO TASKING DESIGN(S)
  - PROVIDES GUIDANCE INTO PACKAGE DESIGN(S)
  - HAS AN ASSOCIATED DESIGN METHODOLOGY THAT IS COMPATIBLE TO Ada

INSTRUCTOR NOTES

IF THE CLASS HAS A GOOD Ada BACKGROUND, DISCUSS THIS SLIDE IN DEPTH, IF NOT, SKIM OVER.

Ada RELATIONSHIP

	IDENTIFIES DATA TYPES	MAPS TO SUBPROGRAMS	GUIDANCE INTO TASKING DESIGN	GUIDANCE INTO PACKING DESIGN	ASSOCIATED DESIGN METHODOLOGY
SADT	M	M	L	L	H
SREM	H	H	M	L	M
ENTITY DIAGRAMMING	M				
PSL/PSA	H	L	L	L	
DEMARCO DATA FLOW	M	M			M
GANE AND SARSON DIAGRAMS	M	M			M
SCRP	H	H	M	M	H

LEVEL OF SUPPORT
H - HIGH
M - MODERATE
L - LOW
BLANK - NO SUPPORT

INSTRUCTOR NOTES

FINISH UP ANALYSIS SECTION BY TRYING TO REVIEW MAIN THREADS PRESENTED IN COURSE. TIE THESE INTO WHAT WILL BE DONE NEXT IN THE DESIGN SECTION.

MISCELLANEOUS Ada RELATIONSHIPS

- USE Ada TO SIMULATE FUNCTIONAL REQUIREMENTS
- MODIFY REQUIREMENTS LANGUAGE ASPECTS OF THE METHODOLOGIES TO (FULL) SYNTAX Ada
  - SREM => RSL
  - PSL/PSA => PSL

INSTRUCTOR NOTES

<u>THEME</u>: DESIGN CONSISTS OF TWO SEPARATE SETS OF ACTIVITIES WHICH TAKE THE OUTPUT OF THE ANALYSIS (AND SPECIFICATION) PHASE OF THE LIFE CYCLE AND CREATES A "PAPER" MODEL OF THE SOFTWARE.

<u>PURPOSE</u>: TO PROVIDE INSIGHT INTO THE MAJOR ASPECTS OF THE DESIGN PHASE OF THE LIFE CYCLE.

<u>REFERENCES</u>: WEINBERG, G., "RETHINKING SYSTEMS ANALYSIS AND DESIGN" LITTLE, BROWN, MA; 1982

Section 13

DESIGN INTRODUCTION

INSTRUCTOR NOTES

DESIGN IS A BLUEPRINT OF MODULES AND THEIR INTERCONNECTIONS. THE DESIGN PROCESS ALLOCATES THE FUNCTIONAL REQUIREMENTS TO A DESIGN STRUCTURE, PRESENTING THE FORM OR DESIGN STRUCTURE ALONG WITH SOME INDICATION OF WHERE THE VARIOUS FUNCTIONS ARE TO BE PERFORMED. THE STRUCTURE AND ALLOCATION SHOULD ALLOW THE PERFORMANCE AND ANALYTIC REQUIREMENTS TO BE FACTORED IN AND VERIFIED AS CONSTRAINTS.

THE INTERFACES BETWEEN THE COMPONENTS OF THE STRUCTURE ARE ALSO DEFINED AT THIS TIME. (MORE EMPHASIS IS PLACED ON THESE INTERFACES (E.G. PARNAS) THAN OTHERS (E.G. STRUCTURED DESIGN).

THE ACTIVITY OF DESIGN IS A <u>MODELING</u> ACTIVITY. A DESIGN LEAVES CERTAIN DETAILS OUT UNTIL LATER. THERE IS A NEED TO BE ABLE TO COMPREHEND A LARGE AMOUNT OF THE SYSTEM TO BE SURE THAT THE GOALS OF THE PARTICULAR DESIGN STRATEGY (WHICH VARY) ARE BEING MET. DURING LATER STAGES OF THE DESIGN PROCESS DETAIL WILL BE ADDED AS WE EXAMINE SMALLER PARTS OF THE DESIGN.

THE DIFFERENT MODELING TECHNIQUES WE WILL COVER ENCOURAGE US TO LEAVE OUT DIFFERENT THINGS. LEAVING OUT THE DETAILS IS THE HARDEST PART OF THE DESIGN PROCESS.

WHAT IS DESIGN?

- DESIGN ...
  - <u>TRANSLATES</u> REQUIREMENTS SPECIFICATIONS INTO A <u>BLUEPRINT</u> OF THE SYSTEM
  - IS A <u>MODEL</u> OF THE SOFTWARE
  - IS DONE BY <u>DESIGNERS</u>

- FOR OUR PURPOSE DESIGN CONSISTS OF TWO MAJOR SUBPHASES
  - ARCHITECTURAL DESIGN (PRELIMINARY DESIGN)
  - DETAILED DESIGN

INSTRUCTOR NOTES

DESIGNERS DEAL WITH MORE OF THE SYSTEM AT ONE TIME THAN IMPLEMENTERS. THEY CONCENTRATE ON GLOBAL ISSUES, POSTPONING DECISIONS HAVING ONLY LOCAL SCOPE. THE DESIGNER PARTITIONS THE SYSTEM IN A MANNER THAT HIDES DECISIONS LIKELY TO CHANGE SEPARATELY INTO INDIVIDUAL MODULES.

CLEAN INTERFACES REALLY MEAN FULLY DEFINED INTERFACES. THE SIMPLER THESE INTERFACES ARE - FEWER OPTIONS, FEWER PARAMETERS - THE MORE LIKELY IT IS TO BE FOR THEM TO BE FULLY DEFINED. COMPLEX INTERFACES PROBABLY MEAN TOO MANY FUNCTIONS IN A SINGLE MODULE.

THE DESIGNER COMMUNICATES WITH THE ANALYST EITHER VERBALLY OR BETTER YET, THROUGH DOCUMENTATION TO MAP FUNCTIONAL REQUIREMENTS (AS WELL AS OTHERS SUCH AS RELIABILITY, MODIFIABILITY, ETC.). THE DESIGNER WORKS WITH IMPLEMENTERS TO ASSESS PERFORMANCE AND OTHER GOALS IN ORDER TO MAKE TRADE-OFFS AMONG THEM.

THE INTERFACE BETWEEN DESIGNERS AND ANALYSTS IS PROBABLY MORE DIFFICULT THAN BETWEEN DESIGNERS AND IMPLEMENTERS. DESIGNERS USUALLY WERE IMPLEMENTERS ONCE AND UNDERSTAND THE AREA WELL.

DESIGNER'S ROLE

- DESIGERS ...
  - POSTPONE IMPLEMENTATION DECISIONS
  - HIDE THEIR DECISIONS INSIDE MODULES
  - WORRY ABOUT THE SOFTWARE'S STRUCTURE
  - SPECIFIES ALGORITHMS AND CONTROL FLOW
  - MAKE CLEAN INTERFACES
  - COMMUNICATE WITH ANALYSTS AND IMPLEMENTERS

- THE DESIGNER IS THE BRIDGE BETWEEN ANALYSTS AND IMPLEMENTERS

INSTRUCTOR NOTES

LET'S LOOK BRIEFLY AT THESE FOUR TECHNIQUES, FOR THEY APPEAR IN ALMOST EVERY METHOD IN SOME FORM OR ANOTHER ...

THESE FOUR TECHNIQUES ARE REALLY JUST GOOD ENGINEERING JUDGMENT (OR COMMON SENSE). THEY ARE HOW WE TACKLE ANY COMPLEX PROBLEM. REMEMBER, THESE WERE ALSO USED BY ANALYSTS.

ARCHITECTURAL DESIGN TECHNIQUES

- DESIGNERS ALSO APPLY SEVERAL TECHNIQUES ...
  - ABSTRACTION (1)
  - MECHANISM (1)
  - ITERATION (2)
  - DECOMPOSITION (1)

(1) STRUCTURING PRINCIPLE

(2) DESIGN PROCESS

INSTRUCTOR NOTES

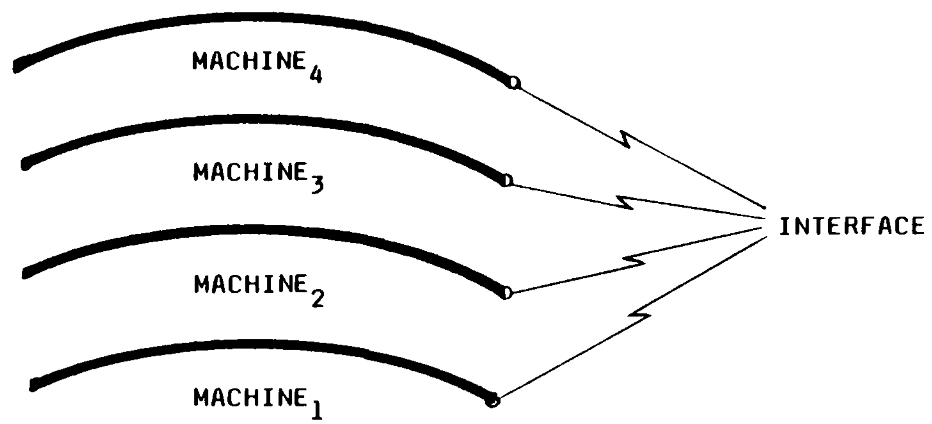
THESE MACHINES ARE OFTEN CALLED "ABSTRACT MACHINES." ONE WAY TO DISCUSS ABSTRACTION IS THAT IT PERMITS MULTIPLE LEVELS OF UNDERSTANDING OF A SYSTEM. THE ISO STANDARD ON COMMUNICATION PROTOCOLS ILLUSTRATES THIS CONCEPT.

INSIDE A MACHINE = SIMILARITIES. THE SIMILARITIES EXIST IN THE DATA STRUCTURES SHARED BY AND USED TO COMMUNICATE WITH PRIMITIVES.

DIFFERENCES ARE EXPRESSED THROUGH INTERFACES. DIFFERENCES ARE VERY OFTEN IN THE MEANS OF COMMUNICATION BETWEEN LAYERS (INVOCATION).

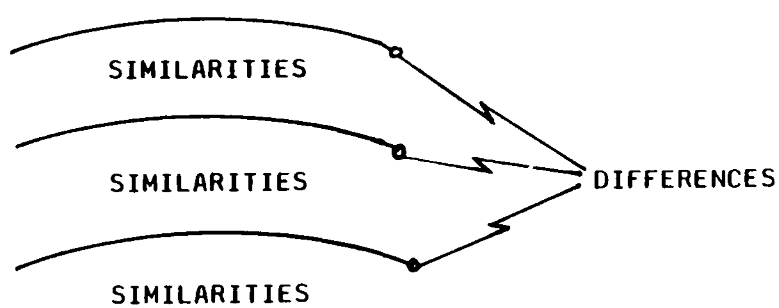
ARCHITECTURAL DESIGN TECHNIQUES

- ABSTRACTION FORMS LAYERS OF INDEPENDENT MACHINES, EACH COMMUNICATING TO EACH OTHER VIA EXPLICIT INTERFACES:



EXAMPLE: COMMUNICATION SOFTWARE (7 LAYER OPEN SYSTEM INTERCONNECTION MODEL)

- ABSTRACTION HELPS SEPARATE DIFFERENCES FROM SIMILARITIES:



INSTRUCTOR NOTES

GOOD DESIGNS HAVE SEVERAL "JUNCTORS."

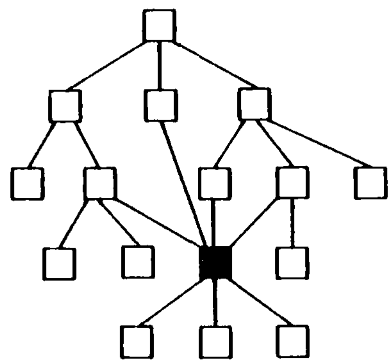
USUALLY (THOUGH NOT ALWAYS) THE MORE THE BETTER.

"MECHANISM" IS ALSO KNOWN AS "SHARED SERVICE" - REALLY JUST AN APPLICATION SPECIFIC INSTANCE OF WHAT HAS LED TO DEVELOPMENT OF GENERAL PURPOSE OPERATING SYSTEM FUNCTIONS. GOOD IDENTIFICATION OF MECHANISMS RESULT IN REDUCED COSTS, HOWEVER, IT DOES TRADE-OFF AGAINST EFFICIENCY AND MODIFIABILITY.

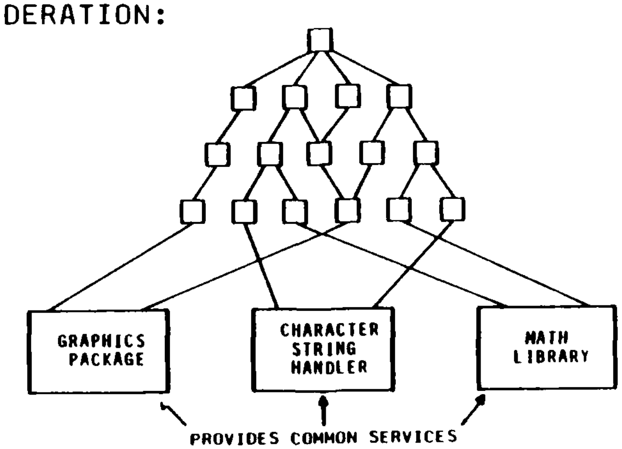
TYPICAL MECHANISMS ARE GRAPHICS, CHARACTER STRING, AND MATH ROUTINE PACKAGES.

ARCHITECTURAL DESIGN TECHNIQUES

- <u>MECHANISM</u> PRODUCES SUBSTRUCTURES THAT SUPPORT MANY AREAS OF THE WHOLE DESIGN:



- MECHANISMS ARE PARTICULAR SUBJECTS, ISOLATED FOR SEPARATE (AND SPECIAL) CONSIDERATION:



INSTRUCTOR NOTES

IDEAS <u>CYCLE</u> DURING DESIGN, SPAWNING MORE IDEAS. A GOOD DESIGN IS VERY RARELY ACHIEVED THE FIRST TIME.

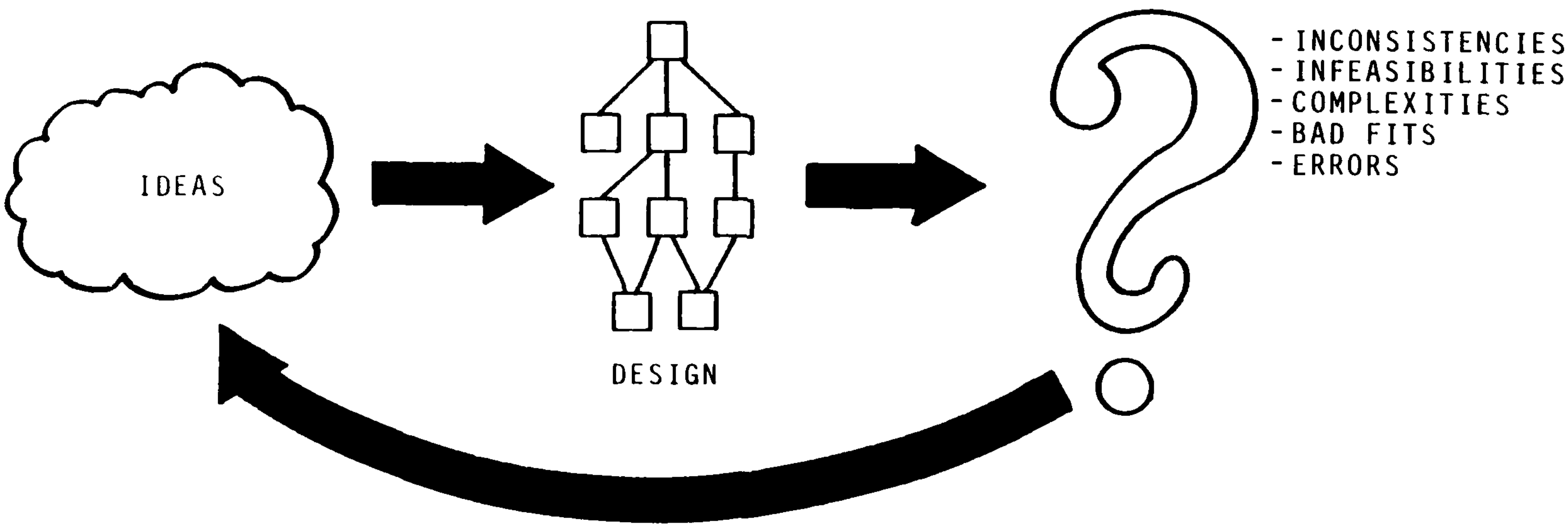
ITERATION STRESSES THE ROLE THAT THE DESIGNER PLAYS IN BRIDGING THE GAP BETWEEN THE ANALYST AND IMPLEMENTER. THERE ALWAYS ARE UNREASONABLE OR CONFLICTING GOALS, AS WELL AS VARIOUS IMPLEMENTATION CHOICES. ITERATION HELPS RESOLVE THE CONFLICTS AND TO MAKE GOOD IMPLEMENTATION SELECTIONS.

PROTOTYPE - "MINIATURE" SYSTEM THAT WORKS WELL ENOUGH TO VERIFY SOME CRITICAL ASPECT OF THE DESIGN.

BOTTOM-UP - VERIFIES LOW-LEVEL DESIGN ASSUMPTIONS, SO IMPACT TO HIGHER LEVELS FOR THE DESIGN IS LOW.

ARCHITECTURAL DESIGN TECHNIQUES

- <u>ITERATION</u> WORKS IN NEW IDEAS, AND WORKS OUT INFEASIBLE IDEAS:



- GOOD DESIGNERS <u>ALWAYS</u> ITERATE. FOR EXAMPLE ...

  - USE OF PROTOTYPES

  - USE OF BOTTOM-UP DESIGN

  VERIFY DESIGN DECISIONS, SO BAD IDEAS CAN BE WORKED OUT <u>BEFORE</u>

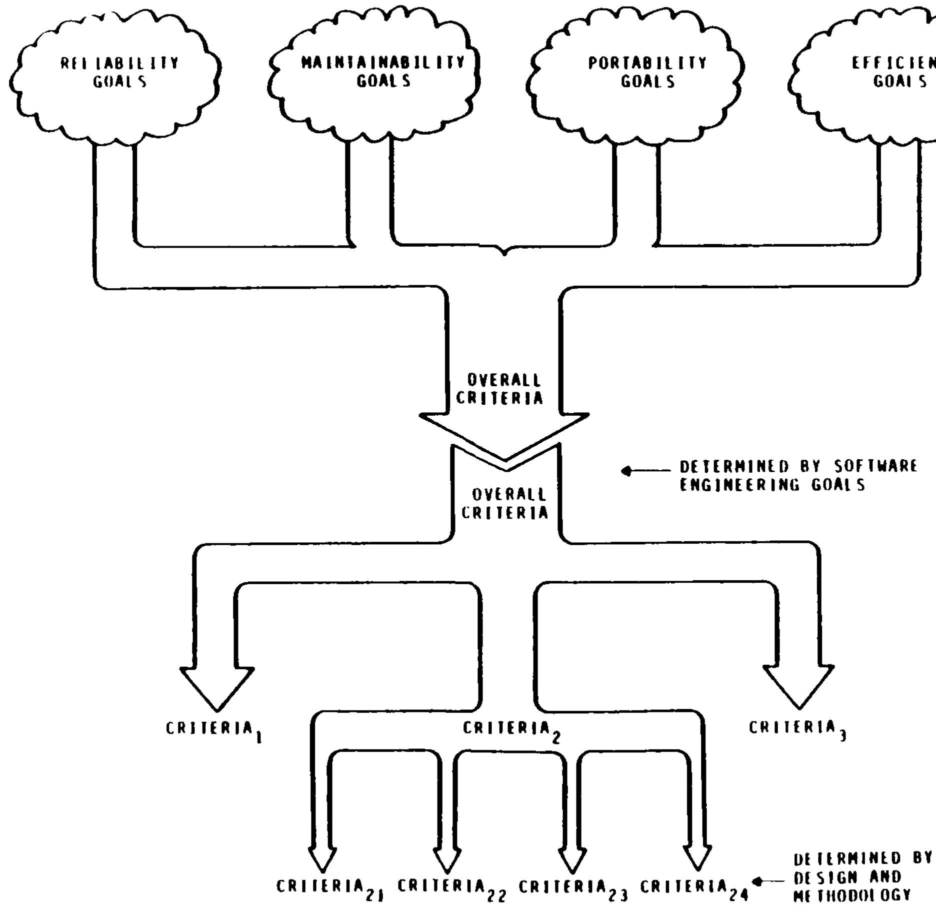
  SYSTEM INSTALLATION

INSTRUCTOR NOTES

DECOMPOSITION CRITERIA MUST NOT FALL OUTSIDE THE CONTEXT OF ANY HIGHER-LEVEL CRITERIA. THIS WAY, ALL PARTS OF THE SYSTEM WORK TOGETHER; NOT ACROSS PURPOSES. DECOMPOSITION BREAKS THINGS DOWN INTO MANAGEABLE PIECES.

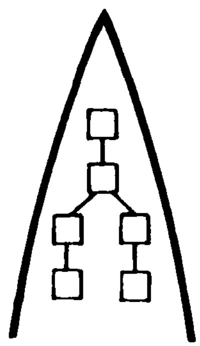
ALL THE SOFTWARE ENGINEERING GOALS (AND THEIR TRADEOFFS) CONTRIBUTE TO FORMULATING DECOMPOSITION CRITERIA.

- DECOMPOSITION REQUIRES CONSISTENTLY APPLIED CRITERIA THAT COMES FROM MANY SOURCES:

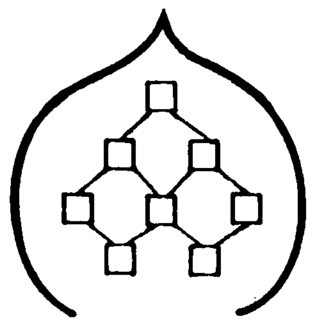


INSTRUCTOR NOTES

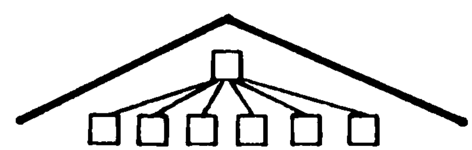
IF YOU WANT TO HAVE A GOOD ARGUMENT ON REQUIREMENTS VERSUS DESIGN JUST VISIT AN ORGANIZATION WHERE ONE GROUP DOES ONE AND ANOTHER DOES THE OTHER. "WHAT" VS "HOW" SOUNDS SIMPLE BUT IT IS VERY HARD TO GET SPECIFIC ABOUT WHAT SOMETHING DOES WITHOUT PROPOSING A HOW. ADDING TO THE DIFFICULTIES IS THE FACT THAT PEOPLE GO FROM IMPLEMENTER TO DESIGNERS TO ANALYSTS. THEY HAVE A HARD TIME FORGETTING WHAT THEY PREVIOUSLY DID. THE DESIGN "BLUEPRINT" MUST DEPICT THE <u>OVERALL</u> STRUCTURE:



TOWER
(BAD)



MOSQUE
(GOOD)



PANCAKE
(BAD)

SOME MEANS OF JUDGING OVERALL STRUCTURE IS IMPORTANT - GRAPHICS ARE NICE AND PREFERRED BUT OTHER METHODS ARE ACCEPTABLE (E.G., INDENTED HIERARCHICAL DIAGRAM).

GENERAL GUIDELINES FOR ARCHITECTURAL DESIGN

- THE BOUNDARY BETWEEN REQUIREMENTS AND DESIGN IS UNCLEAR (AND ALWAYS WILL BE)
  - IMPLIES ITERATION BETWEEN ANALYSIS AND DESIGN
- <u>NO SINGLE</u> METHOD GIVES A COMPLETE PROCESS ON HOW TO GO FROM REQUIREMENTS TO DESIGN
  - SOFTWARE COST REDUCTION PROJECT (SCRP) COMES CLOSE
- ALWAYS BUILD A PICTURE OF THE SOFTWARE ARCHITECTURE
  - MAKE ARCHITECTURE VISIBLE
  - SHOW INTERRELATIONSHIP BETWEEN COMPONENTS THAT MAKE UP ARCHITECTURE
- MAKE YOUR GOAL A SET OF DELIVERABLES

INSTRUCTOR NOTES

THESE ARE THE ELEMENTS REQUIRED BY SDS. EACH DELIVERED ITEM MUST BE TECHNICALLY ADEQUATE, UNDERSTANDABLE AND COMPREHENSIVE.

DoD-STD-SDS VIEW OF ARCHITECTURAL DESIGN

- SDS PRELIMINARY DESIGN ACTIVITIES
  - ASSIGN FUNCTIONAL AND PERFORMANCE SOFTWARE REQUIREMENTS TO TOP LEVEL SOFTWARE COMPONENTS
  - USE TOP-DOWN DESIGN APPROACH
  - ESTABLISH TOP-LEVEL DESIGN
    - DETERMINE FLOW OF DATA AND EXECUTION OF CONTROL BETWEEN COMPONENTS
    - SPECIFY TOP-LEVEL ALGORITHMS
  - TOP-LEVEL DESIGN SHALL USE A PROGRAM DESIGN LANGUAGE (PDL)
  - DEVELOP TEST PLANS
  - DEVELOP USER MANUALS (IF NEEDED)

INSTRUCTOR NOTES

SDS CONTINUED ...

DoD-STS-SDS VIEW OF ARCHITECTURAL DESIGN

- SDS PRELIMINARY DESIGN PRODUCTS
  - RECORDS OF REVIEWS, ACTION ITEMS, ETC.
  - SOFTWARE TOP-LEVEL DESIGN DOCUMENT
  - SOFTWARE TEST PLAN
  - COMPUTER SYSTEM OPERATOR'S MANUAL, SOFTWARE USER'S MANUAL, COMPUTER DIAGNOSTIC MANUAL *(IF NEEDED)*

- SDS PRELIMINARY DESIGN REVIEWS
  - A PRELIMINARY DESIGN REVIEW
    - REVIEWS DESIGN PRODUCTS
    - DEMONSTRATION OF TRACEABILITY BACK TO THE REQUIREMENTS

INSTRUCTOR NOTES

STRESS THAT DETAILED DESIGN IS A <u>TRANSITIONARY</u> ACTIVITY.

(a) REVIEW THE DIFFERENCES BETWEEN REQUIREMENT AND SPECIFICATIONS ON ONE HAND (i.e., DETERMINE WHAT IS TO BE DONE) AND DESIGN ON THE OTHER (i.e., HOW IT IS TO BE DONE, THE ANALYSIS OF VARIOUS ALTERNATIVES, AND THE TRADE-OFFS MADE BASED ON THE CONSTRAINTS SPECIFIED).

(b) DESIGN ACTIVITY IS DIVIDED INTO 2 DISTINCT PHASES: ARCHITECTURAL AND DETAILED:

(1) ARCHITECTURAL EMPHASIZES STRUCTURE OF THE SYSTEM, THE DECOMPOSITION OF THE SYSTEM INTO MODULES, AND THE PRECISE SPECIFICATION OF THE INTERFACES BETWEEN THEM.

(2) DETAILED DESIGN EMPHASIZES THE SELECTION AND EVALUATION OF THE ALGORITHMS NECESSARY TO CARRY OUT THE LOGICAL STEPS SPECIFIED FOR INDIVIDUAL MODULES.

(c) DESIGN IS A VERY ITERATIVE PROCESS.

ASPECTS OF DETAILED DESIGN

- DETAILED DESIGN SPANS THE GAP FROM ARCHITECTURAL DESIGN TO CODE
- THERE IS CONTROVERSY ABOUT WHEN DETAILED DESIGN BEGINS AND ENDS AND WHAT FORM IT TAKES
- BUT EVERYONE AGREES THAT DETAILED DESIGN PRODUCES SPECIFICATIONS SUFFICIENT <u>TO CODE FROM</u>

INSTRUCTOR NOTES

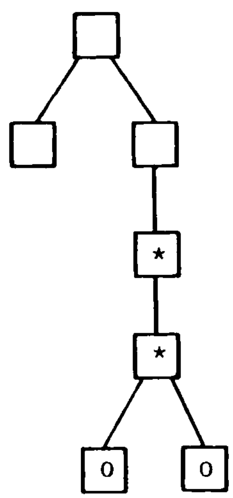
THE LINES ARE EVEN FUZZIER BETWEEN DETAILED DESIGN AND ARCHITECTURAL DESIGN AND CODING, THAN BETWEEN REQUIREMENTS AND ARCHITECTURAL DESIGN.

THE "CONTROVERSY" EXISTS PRECISELY BECAUSE DESIGN IS AN ITERATIVE PROCESS. BOTH DETAILED AND ARCHITECTURAL DESIGN SERVE TO TIGHTEN UP ANY LOOSE POINTS IN THE SPECIFICATIONS, AND HELPS DEFINE FOR THE PROGRAMMER THE STEPS NEEDED TO BE INCLUDED IN THE ACTUAL IMPLEMENTATION. ITERATION BETWEEN THE SUB-PHASES AFFECTS THE DISTINCTION BETWEEN THE TWO.

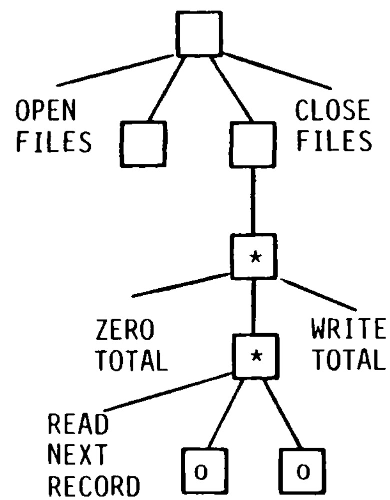
TRANSITION FROM ARCHITECTURAL TO DETAILED DESIGN

- MOST ARCHITECTURAL DESIGN METHODOLOGIES HAVE A WAY OF GOING FROM "BLUEPRINT" TO CODE SPECS. FOR EXAMPLE ...

- JACKSON USES "COMPLETION", ADDING PSEUDO-CODE TO APPROPRIATE PLACES ON THE STRUCTURE CHART:



STRUCTURE CHART



PROGRAM SPECIFICATION

INSTRUCTOR NOTES

OFTEN, THIS TRANSITION IS NOT PART OF A FORMAL METHODOLOGY.

DETAILED DESIGN METHODOLOGIES

- MOST ARCHITECTURAL DESIGN METHODS STOP WHEN THE MODULE OR UNIT LEVEL IS REACHED

- SO MANY DESIGN METHODOLOGIES ADDRESS DESIGN ISSUES <u>INSIDE</u> A MODULE EMPHASIZING ONE OR MORE OF THE FOLLOWING:
  - PHYSICAL STORAGE (FILES, DATABASES, ETC.)
  - LOGIC FLOW
  - ALGORITHM
  - DATA STRUCTURE

INSTRUCTOR NOTES

AGAIN, EACH DELIVERED ITEM MUST BE TECHNICALLY ADEQUATE, UNDERSTANDABLE, AND COMPREHENSIVE.

DoD-STD-SDS VIEW OF DESIGN

- SDS DETAILED DESIGN ACTIVITIES
  - <u>REFINE</u> TOP-LEVEL SOFTWARE COMPONENTS TO LOWER-LEVELS
    - PERFORM UNTIL A SINGLE, NON-DIVISBLE FUNCTION IS DEFINED
  - USE A TOP-DOWN APPROACH
  - USE BOTTOM-UP APPROACH ONLY ON CRITICAL LOWER-LEVEL UNITS (I.E. COMMON ROUTINES, EXECUTIVE, ETC.)
  - USE A PROGRAM DESIGN LANGUAGE
  - ESTABLISH SOFTWARE DEVELOPMENT FOLDERS FOR ALL SOFTWARE UNITS CONTAINING:
    - UNIT REQUIREMENTS
    - DESIGN INFORMATION
    - CODE
    - TEST PROCEDURES
    - UNIT SCHEDULE
    - STATUS

SDS DESIGN VIEW CONTINUED ...

DoD-STD-SDS VIEW OF DESIGN

  - DEVELOP TEST FOR EACH UNIT
  - DEVELOP INTEGRATION TEST CASES

- SDS DETAILED DESIGN PRODUCTS
  - SOFTWARE DETAILED DESIGN DOCUMENT
  - INTERFACE DESIGN DOCUMENT
  - DATABASE DESIGN DOCUMENT
  - SOFTWARE TEST DESCRIPTION
  - SOFTWARE DEVELOPMENT FOLDER
  - SOFTWARE PROGRAMMER'S MANUAL*
  - FIRMWARE SUPPORT MANUAL*

- SDS DETAILED DESIGN REVIEWS
  - CRITICAL DESIGN REVIEW OF PRODUCTS ABOVE

---

*PRODUCED IF REQUIRED BY PROCURING AGENCY

INSTRUCTOR NOTES

THESE ARE THE MAJOR VIEWPOINTS AND FORMATS USED IN MOST DESIGN PROCESSES.

DESIGN METHODOLOGY PERSPECTIVES AND FORMATS

- PRIMARY PERSPECTIVES CONSIDERED DURING DESIGN:
  - DATA-FLOW: FOCUSES ON THE FLOW OF DATA BETWEEN FUNCTIONS
  - DATA STRUCTURE: FOCUSES ON DETERMINING PROGRAM STRUCTURE FROM RELATIONSHIPS OF DATA ELEMENTS
  - OTHER: FOCUS DETERMINED BY A SET OF FUNDAMENTAL CONCEPTS
    - INFORMATION HIDING
    - ABSTRACT DATA TYPES
    - AXIOMATIC MATHEMATICS
    - ETC.
  - PROCEDURAL: FOCUS ON SPECIFYING ALGORITHM AND CONTROL FLOW

- PRIMARY FORMATS
  - GRAPHICAL
  - STRUCTURED LANGUAGE
  - STRUCTURED TEXT

INSTRUCTOR NOTES

REVIEW THE DEFINITION OF "OTHER" FROM PREVIOUS SLIDE AND FILL IN THE TYPE(S) USED, E.G. HOS USES AXIOMATIC PROOFS OF CORRECTNESS.

DESIGN METHODOLOGIES TO BE PRESENTED

ARCHITECTURAL

METHODOLOGY	PERSPECTIVE	FORMAT
• SOFTWARE COST REDUCTION PROJECT (SCRP)	• OTHER	• STRUCTURED LANGUAGE
• OBJECT-ORIENTED DESIGN	• OTHER	• GRAPHICAL • STRUCTURED TEXT
• STRUCTURED DESIGN	• DATA FLOW	• GRAPHICAL
• JACKSON DESIGN	• DATA STRUCTURE	• GRAPHICAL • STRUCTURED TEXT
• WARNIER-ORR	• DATA STRUCTURE	• STRUCTURED TEXT • GRAPHICAL
• HIGHER ORDER SOFTWARE	• OTHER	• GRAPHICAL

DETAILED

• HIPO	• DATA FLOW • PROCEDURAL	• STRUCTURED TEXT • GRAPHICAL
• NASSI-SCHNEIDERMAN	• PROCEDURAL	• GRAPHICAL
• PROGRAM DESIGN LANGUAGE	• DATA STRUCTURE • PROCEDURAL	• STRUCTURED LANGUAGE

INSTRUCTOR NOTES

TELL THE CLASS THESE WILL BE DISCUSSED LATER IN THE COURSE.

OTHER AREAS TO BE COVERED

- QUALITY MEASUREMENT AND ASSESSMENT TECHNIQUES
  - COUPLING
  - COHESION
  - DESIGN HEURISTICS

INSTRUCTOR NOTES

THEME: IN THIS SECTION WE WILL BE TALKING ABOUT THE DESIGN ASPECTS OF THE SOFTWARE COST REDUCTION (SCR) METHODOLOGY. THIS IS A METHODOLOGY THAT HAS EVOLVED FROM SOME WELL DEVELOPED UNDERLYING CONCEPTS (I.E. MODULARITY, ABSTRACTIONS) AND FROM THE APPLICATION OF THESE CONCEPTS TO A REALISTIC, REAL TIME EMBEDDED SYSTEM. WE WILL COVER BOTH THE UNDERLYING CONCEPTS AND PRESENT DESIGN PRINCIPLES FROM THE ACTUAL PROJECT (THE REDEVELOPMENT OF THE A-7E AIRCRAFT *OPERATIONAL FLIGHT PROGRAM)* THAT ARE APPLICABLE TO A WIDE RANGE OF DoD SYSTEMS.

PURPOSE: TO PROVIDE AN OVERVIEW OF AN EVOLVING DESIGN METHODOLOGY ORIENTED TOWARD DoD APPLICATION.

REFERENCES: PARNAS, D. "ON THE CRITERIA TO BE USED IN DECOMPOSING SYSTEMS INTO MODULES" CACM VOL. 15, NO. 12; DECEMBER 1972.

Section 14

SOFTWARE COST REDUCTION PROJECT DESIGN METHODS

INSTRUCTOR NOTES

PARNAS IS ONE OF THE MORE VAGUELY DEFINED METHODOLOGIES, NOT IMPLYING THAT IT IS ANY LESS IMPORTANT THAN THE OTHERS. HOWEVER, THERE IS NO SINGLE BOOK OR PRODUCT OF A COMPANY THAT EXISTS THAT FULLY DESCRIBES IT. OBJECT-ORIENTED DESIGN IS TO SOME EXTENT A SPECIFIC REFINEMENT OF PARNAS' IDEAS APPLIED SPECIFICALLY TO Ada.

ALL OF THESE CHARACTERISTICS HAVE BEEN ESPOUSED FOR MANY YEARS BY MANY OF THE PROMINENT NAMES OF SOFTWARE ENGINEERING; WIRTH, DIJKSTRA, PARNAS, ETC. WE WILL COVER EACH OF THE CHARACTERISTICS IN MORE DETAIL. THE SCR PROJECT PRESENTS A DOCUMENTED, WORKED OUT EXAMPLE OF THE APPLICATION OF THESE DESIGN PRINCIPLES. THE SCR PROJECT IS THE ONLY NON-TRIVIAL APPLICATION OF THESE PRINCIPLES.

OVERVIEW

- SOFTWARE COST REDUCTION PROJECT (SCRP) DESIGN METHODS ARE BASED ON THE WORK OF D. PARNAS AT NRL

- THE GOAL OF THIS METHOD IS TO DEVELOP A SPECIFICATION OF MODULES, HAVING GOOD DESIGN CHARACTERISTICS ...
  - USE ABSTRACTION, SOMETIMES LEVELS OF ABSTRACTION
  - ARE MODULAR
  - HIDE DESIGN DECISIONS

INSTRUCTOR NOTES

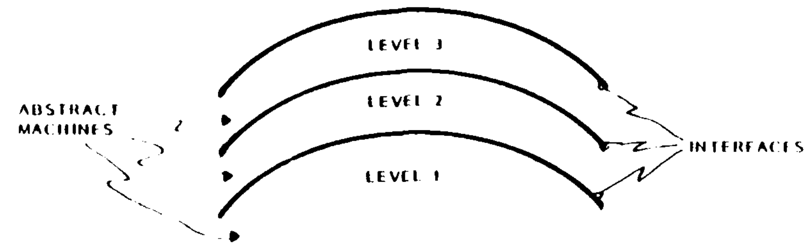
THE EMPHASIS IS ON ABSTRACTION - NOT ON LEVELS. NEITHER PARNAS NOR OBJECT-ORIENTED DESIGN FUSS TOO MUCH WITH LAYERS OR HIERARCHICAL STRUCTURING. THESE ARE USED WHERE APPROPRIATE BUT REALLY THIS TECHNIQUE IS FLATTER THAN SOME OTHERS.

ABSTRACTION, PARTICULARLY AS IT RELATES TO BUILDING UP A SYSTEM THROUGH LEVELS OF ABSTRACT MACHINES WAS POPULARIZED BY DIJKSTRA. THE MOST IMPORTANT PAPER BEING ON THE "THE" SYSTEM, A GENERAL PURPOSE OPERATING SYSTEM.

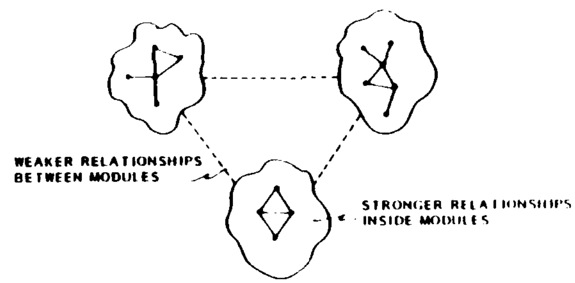
MODULARITY IS PROBABLY THE OLDEST CONCEPT AND THUS HARD TO IDENTIFY AS BEING FATHERED BY ANY ONE PERSON. EVERYONE AGREES THAT MODULARITY IS GOOD, ALTHOUGH SOME STILL FEEL THAT EXCESSIVE MODULARITY WILL CREATE PERFORMANCE PROBLEMS. THE QUESTION ISN'T REALLY WHETHER OR NOT TO USE MODULARITY OR NOT, BUT HOW TO DECIDE ON THE RIGHT MODULE PARTITIONING. HIDING IS ONE POTENTIAL STRATEGY FOR DECIDING ON A PARTICULAR MODULAR DECOMPOSITION. PARNAS HAS WRITTEN SEVERAL EXCELLENT PAPERS ON THE SUBJECT OF HIDING AS IT RELATES TO MODULARITY AND ABSTRACTION.

OVERVIEW

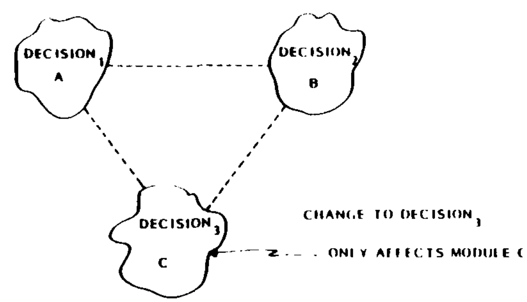
- ABSTRACTION CREATES LEVELS OF SOFTWARE, EACH BUILDING AND RELYING UPON THE OTHER: NONE KNOWING THE WORKINGS OF THE OTHER ...



- MODULARITY BREAKS A SYSTEM INTO PIECES, EACH PERFORMING CLOSELY-RELATED FUNCTIONS ...



- HIDING DESIGN DECISIONS INSIDE MODULES LIMITS THE IMPACT OF CHANGE ...



INSTRUCTOR NOTES

WE WILL DISCUSS ABSTRACTION FIRST, AS A CONCEPT INDEPENDENT OF SOFTWARE OR MODULARITY, WE WILL THEN PRESENT AN EXAMPLE OF DIFFERENT MODULARIZATIONS OF A SMALL SYSTEM. THIS EXAMPLE SHOWS HOW DIFFERENT DESIGN DECISIONS WILL BE HIDDEN BY THE TWO MODULARIZATIONS AND GIVE GUIDANCE ON HOW TO DECIDE WHAT TO HIDE (I.E., A MODULARIZATION STRATEGY).

KEY CONCEPTS

- SYSTEMS CAN BE BUILT USING SMALL, INDEPENDENT MODULES
- EACH MODULE PRESENTS TO THE OUTSIDE WORLD AN ABSTRACT INTERFACE
- DESIGN DECISIONS SHOULD BE HIDDEN INSIDE MODULES
- WE WILL PRESENT THE FOLLOWING CONCEPTS:
  - ABSTRACTION
  - MODULARITY
  - HIDING

INSTRUCTOR NOTES

ABSTRACTION, AS A CONCEPT, CLEARLY PREDATES ITS USE HERE. WE ABSTRACT ABOUT EVERYTHING AROUND US. WE THINK OF A CAR IN TERMS OF ITS GAS PEDAL, BRAKE, STEERING WHEEL, SEATS, ETC. WITHOUT EVER HAVING TO KNOW ANYTHING ABOUT THE DETAILED WORKINGS OF ENGINES OR OTHER SYSTEMS. IN GENERAL WE DON'T HAVE TO UNDERSTAND THE DETAILED COMPOSITION AND WORKINGS OF AN OBJECT IN ORDER TO USE OR INTERACT WITH IT - WE DO THIS BY ABSTRACTING OUT THE PROPERTIES WHICH ARE IMPORTANT TO US.

WE WILL NOW PRESENT AN EXAMPLE OF HOW WE GO ABOUT LEARNING (OR CREATING) AN ABSTRACTION - SOMETHING WE DO EVERY DAY WITHOUT NECESSARILY BEING AWARE OF IT.

ABSTRACTION

- TRADITIONALLY PEOPLE ABSTRACT ABOUT REAL-WORLD THINGS AND SYSTEMS
  - A CAR
  - A CAT
  - A RADAR SYSTEM

- ABSTRACTION IS A PROCESS BY WHICH WE EMPHASIZE THE ESSENTIAL PROPERTIES WHILE SUPPRESSING THE NON-ESSENTIAL DETAILS
  - RECOGNIZING SIMILARITIES AND IGNORING DIFFERENCES

- THE PROCESS OF <u>ABSTRACTION</u> CAN BE SUMMARIZED IN FOUR STEPS:
  1. ABSTRACT
  2. PRESENT
  3. MANIPULATE
  4. POSTULATE

INSTRUCTOR NOTES

THIS IS A FOUR WHEELED VEHICLE. EACH FRONT WHEEL IS CONNECTED TO TWO STRINGS. A DRIVER STEERS THE VEHICLE BY PULLING OR RELEASING ALL FOUR STRINGS SIMULTANEOUSLY. THE OBJECT UNDER STUDY ARE THE WHEELS.

ESSENTIAL PROPERTIES:

- FRONT WHEELS FREE TO ROTATE
- MUST POINT IN SOME DIRECTION FOR CAR TO MOVE AT ALL
- THE STRINGS ARE RELATED - IF WE PULL ONE, THE OTHER MUST BE RELAXED

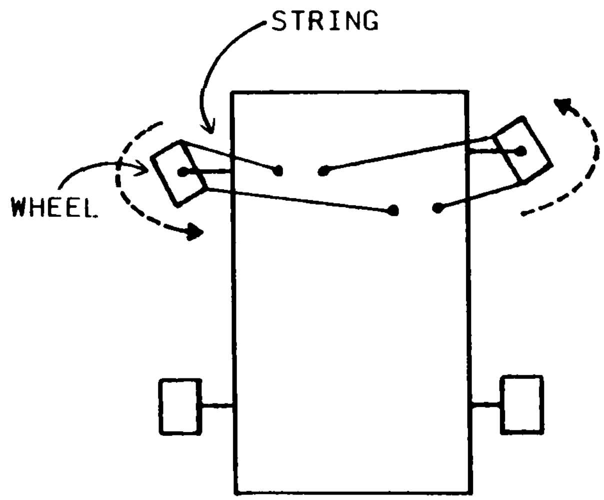
NON-ESSENTIAL DETAILS

- FRONT AND REAR WHEELS ARE THE SAME DISTANCE APART
- ALL WHEELS ARE THE SAME SIZE
- WHEELS ARE PLACED SIDE BY SIDE
- ETC.

ABSTRACTION PROCESS EXAMPLE

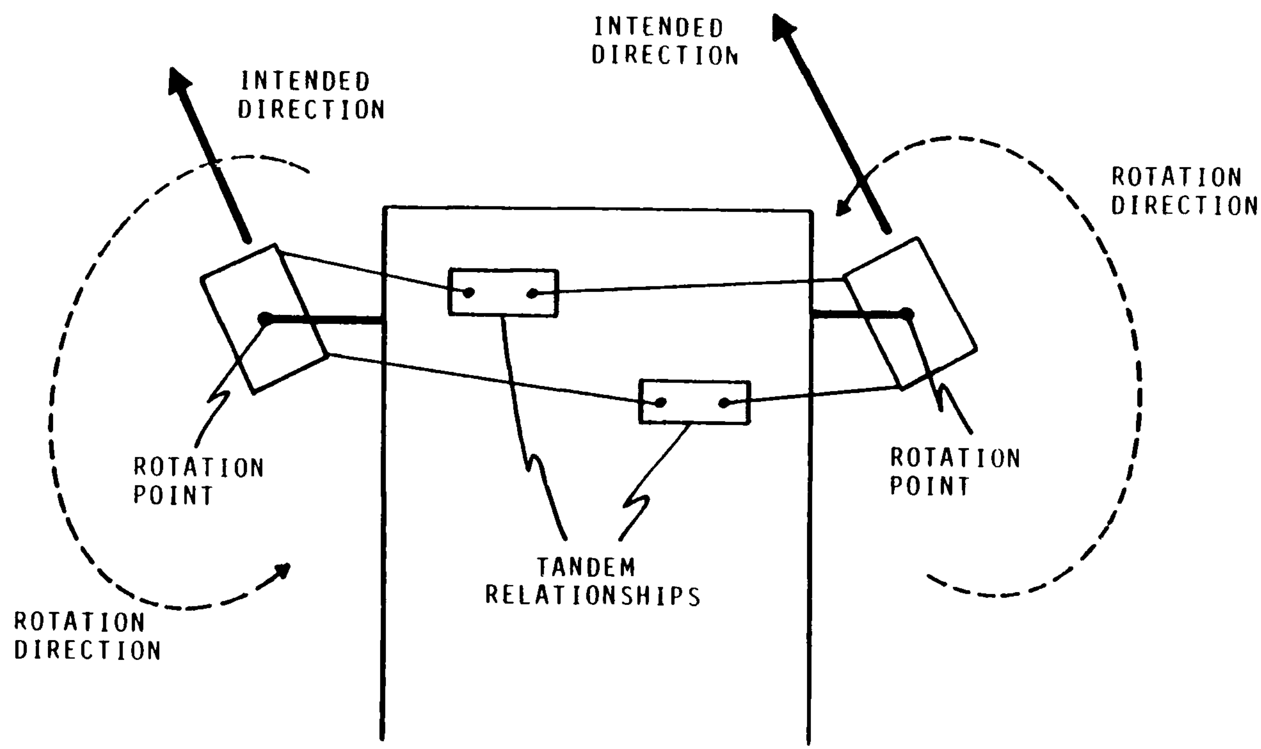
(STEERING SYSTEM OF AN AUTOMOBILE)

(1) <u>ABSTRACT</u> - CONCENTRATE ON THE ESSENTIAL PROPERTIES, WHILE IGNORING THE NON-ESSENTIAL, DETAILS OF A SYSTEM:



- FRONT WHEELS ROTATE SO THEY CAN STEER THE VEHICLE.
- BOTH WHEELS MUST POINT IN THE SAME DIRECTION FOR PROPER OPERATION.
- BOTH FRONT STRINGS AND BOTH BACK STRINGS MOVE IN TANDEM (I.E. WHEN ONE IS PULLED THE OTHER MUST BE RELAXED).

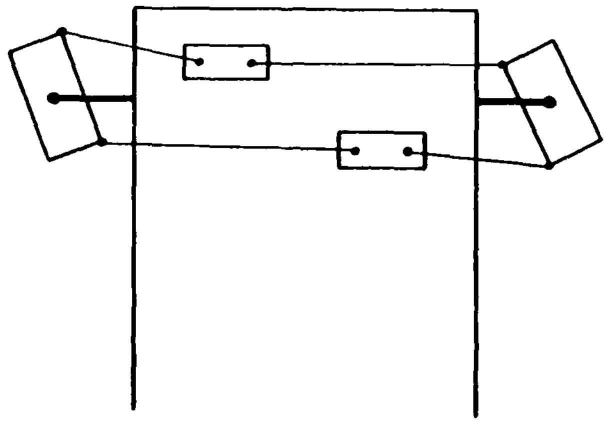
THE BEST NOTATION IS STILL A PICTURE ...



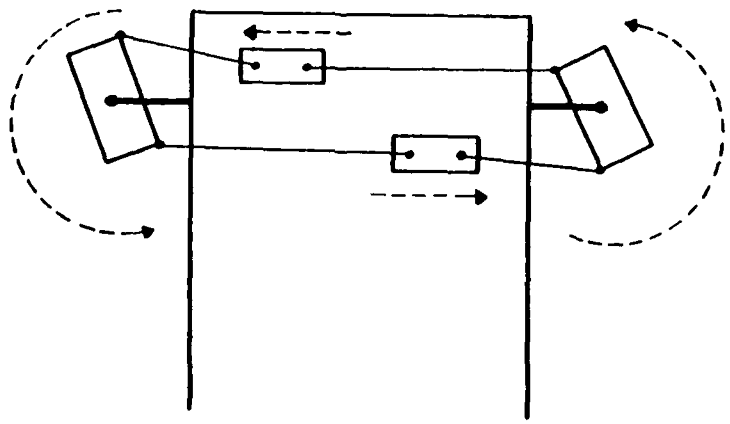
- MOVE TOP BLOCK LEFT AND BOTTOM BLOCK RIGHT TO TURN LEFT
- MOVE TOP BLOCK RIGHT AND BOTTOM BLOCK LEFT TO TURN RIGHT

ABSTRACTION PROCESS EXAMPLE

(2) <u>REPRESENT</u> - FIND A NOTATION THAT SYMBOLIZES THE ABSTRACT CONCEPTS:



(3) <u>MANIPULATE</u> - DEFINE RULES THAT TELL HOW THE SYMBOLS ARE OPERATED ON TO PREDICT THE SYSTEM'S BEHAVIOR:



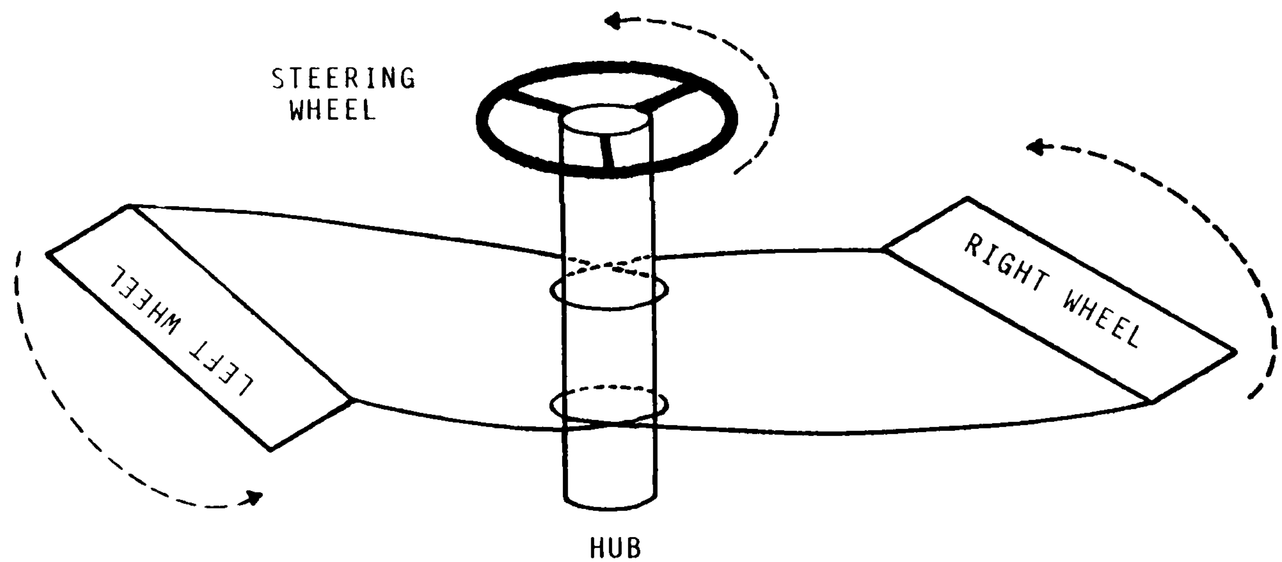
INSTRUCTOR NOTES

NOW WE POSTULATE THE "STEERING WHEEL." THE WHEEL IS ROTATED IN THE DIRECTION WE WISH TO TURN. THE HUB ROTATES CORRESPONDINGLY. THE STRINGS ARE WRAPPED AROUND THE HUB IN THE ONLY WAY TO PROPERLY TRANSLATE TURNING THE STEERING WHEEL INTO ROTATING THE FRONT WHEELS. PROOF IS DONE BY ACTUALLY TURNING THE STEERING WHEEL!

WE COULD HAVE POSTULATED THESE SAME RULES ABOUT THE BLOCKS IN THE PREVIOUS SLIDE.

ABSTRACTION

4 POSTULATE - STATE THE FACTS, PRINCIPLES, OR RULES ABOUT THESE PROPERTIES IN A WAY THAT CAN BE PROVEN CORRECT:



INSTRUCTOR NOTES

NOW, IN GENERAL ...

THE STEERING WHEEL EXAMPLE COULD JUST AS EASILY HAVE BEEN IMPLEMENTED WITH A WORM GEAR OR COMPLETELY WITH ELECTRONIC CONTROLS.

WE USE THE FACT THAT ABSTRACTS WORK ON FAMILIES OF OBJECTS TO LEARN NEW ONES EVERYDAY. WE DON'T RELEARN DRIVING IN EACH NEW CAR. WE EVEN EXPECT A BOAT TO STEER THE SAME AS A CAR BECAUSE THE OBJECTS (BOATS AND CARS) HAVE SAME RELATIONSHIPS. THE MIND AUTOMATICALLY IDENTIFIES WHAT FAMILY A NEW OBJECT FALLS INTO AND EXTENDS TO IT ALL THE PREVIOUS ABSTRACTIONS WE HAVE LEARNED FOR THAT FAMILY.

ABSTRACTION

- AN ABSTRACT MACHINE:
  - DOES NOT DIVULGE HOW IT DOES ITS JOB,
  - PRESENTS AN INTERFACE TO ITS ENVIRONMENT,
  - OPERATES ON A FAMILY OF OBJECTS.

- THE STEERING SYSTEM IS AN ABSTRACT MACHINE ...
  - DOESN'T REQUIRE KNOWLEDGE OF HOW THE STRINGS ARE WRAPPED AROUND THE HUB.
  - PRESENTS A SIMPLE AND SELF-EXPLANATORY INTERFACE TO THE DRIVER.
  - OPERATES ON A FAMILY OF OBJECTS (THE FRONT WHEELS).

INSTRUCTOR NOTES

RE-EMPHASIZE THAT IN THE INITIAL STAGES OF SOFTWARE DESIGN ONE CONCERNS ONESELF WITH THE DESIGN, NOT THE DETAILS. ONE CONCERNS ONESELF WITH THE MOST ESSENTIAL PROPERTIES USING ABSTRACTION AS MUCH AS POSSIBLE. HIGH LEVEL DESIGN SHOULD BE PERFORMED BY REUSING KNOWN OBJECTS FROM OTHER SYSTEMS, POINTING OUT THE DIFFERENCES WHEREVER NECESSARY.

ABSTRACTION

- ABSTRACTION ...
  - FORMS A HIGHER LEVEL OF WORKING, TALKING, ETC.
  - WORKS BOTTOM-UP, BY STARTING WITH A FAMILY OF OBJECTS AND DESIGNING A MACHINE TO MANIPULATE THEM
  - REMOVES TRANSPARENCY BY HIDING HOW THINGS WORK

THE ISSUE THAT WILL NEED TO BE ADDRESSED IS HOW TO DECOMPOSE A SYSTEM CORRECTLY, AND WHETHER SOFTWARE COMPONENTS, OR MODULES, CAN BE "RE-USED," AS IS THE CASE IN HARDWARE DESIGN. THIS IS AN ACTIVE AREA OF RESEARCH TODAY.

THIS IS AN IDEAL VIEW TODAY. TOMORROW IT MAY BE REALITY.

THIS VIEW IS IDEAL BECAUSE MODULARITY ALONE DOES NOT GUARANTEE THAT CHANGES WILL NOT RIPPLE THROUGH A SYSTEM. POORLY MODULARIZED SYSTEMS CAN BE AS BAD AS NON-MODULARIZED SYSTEMS. WE MAY STILL HAVE TO CHANGE MANY MODULES, EVEN FOR A SIMPLE FUNCTIONAL CHANGE AND NOW WE HAVE A HOUSEKEEPING OR CM PROBLEM TO KEEP ALL THE MODULES IN SYNCH. WE NEED A WELL DEVELOPED PURPOSE TO OUR STRUCTURING.

MODULARITY

- MODULARITY DEALS WITH HOW THE STRUCTURE OF AN OBJECT CAN MAKE THE ATTAINMENT OF SOME PURPOSE EASIER
  - THE WAY RELATED PARTS AND PIECES ARE GROUPED TOGETHER
  - PURPOSEFUL STRUCTURING
- MODULARITY RELIES ON DECOMPOSITION AND IS IMPLEMENTED WITH MODULAR DESIGN AND PROGRAMMING
- IN MODULAR DESIGN AND PROGRAMMING ...
  - MODULES CAN BE INDEPENDENTLY CODED, COMPILED, AND TESTED
  - MODULES CAN BE REASSEMBLED OR REPLACED WITHOUT REASSEMBLY OF THE WHOLE SYSTEM

INSTRUCTOR NOTES

THESE BENEFITS ONLY OCCUR IF THE MODULARITY IS "GOOD". IF IT ALLOWS MODULES TO BE DEVELOPED INDEPENDENTLY WITH ANY CHANGES THAT OCCUR DURING DESIGN OR LATER BEING ABLE TO BE HANDLED INTERNAL TO ONE MODULE.

MODULARITY

- <u>EXPECTED</u> BENEFITS ...

  - <u>MANAGERIAL</u> - SHORTER DEVELOPMENT TIME, BECAUSE SEPARATE GROUPS WORK ON EACH MODULE WITH MINIMAL COMMUNICATION

  - <u>FLEXIBILITY</u> - CHANGES CAN BE MADE TO A MODULE WITHOUT CHANGING OTHER MODULES

  - <u>COMPREHENSIBILITY</u> - THE SYSTEM CAN BE STUDIED ONE MODULE AT A TIME

- BENEFITS DON'T COME AUTOMATICALLY BECAUSE OUR DECOMPOSITION CRITERIA IS OFTEN VERY POOR OR CONFLICT WITH EACH OTHER

INSTRUCTOR NOTES

POINT OUT THAT THESE CRITERIA ARE PROBABLY IN CONFLICT - WE CAN'T NECESSARILY MEET THEM ALL AT THE SAME TIME. IN PARTICULAR MAKING UNDERSTANDABLE MODULES MIGHT WELL BE IN CONFLICT WITH ISOLATING ERRORS OR REDUCING THE IMPACT OF CHANGE.

THE SCRP APPROACH SOUNDS GOOD, AND REDUCES THE IMPACT OF FUTURE CHANGES, BUT REQUIRES A LOT OF KNOWLEDGE ABOUT WHAT ASPECTS OF THE SYSTEM (HARDWARE AND PERFORMANCE REQUIREMENTS) ARE LIKELY TO CHANGE.

MODULARITY

- WHAT ARE DETAIL DECOMPOSITION CRITERIA? IN ACTUALITY, ALMOST ANYTHING ...
  - INDEPENDENTLY DEVELOP SUBSYSTEMS
  - REDUCE THE IMPACT OF CHANGES TO REQUIREMENTS
  - UNDERSTAND THE SYSTEM
  - ISOLATE AND CORRECT ERRORS

  .

  .

  .

- SCRP METHODS BASED ON ASSUMPTION THAT A "GOOD" MODULARIZATION RELIES ON HIDING AS MUCH AS POSSIBLE THAT IS LIKELY TO CHANGE AS A UNIT INSIDE A MODULE

WE'RE GOING TO GENERATE ALL CIRCULAR SHIFTS OF A LINE, AND THEN PRINT THEM OUT IN <u>ALPHABETICAL</u> ORDER. ONCE ALPHABETIZED, THE KEY WORDS CAN EASILY BE GENERATED.

EXAMPLE: IF THE TEXT FILE IS:

```
THE CAT IN THE HAT
HORTON HEARS A WHO
THE GRINCH WHO STOLE CHRISTMAS
```

THE KWIC PROGRAM WOULD GENERATE:

1. A WHO HORTON HEARS
2. CAT IN THE HAT
3. CHRISTMAS THE GRINCH WHO STOLE
4. GRINCH WHO STOLE CHRISTMAS THE
5. HAT THE CAT IN THE
6. HEARS A WHO HORTON
7. HORTON HEARS A WHO

.
.
.
.

REFERENCE: "ON THE CRITERIA TO BE USED IN DECOMPOSING SYSTEMS INTO MODULES" D.L. PARNAS 1972.

THIS REFERENCE DESCRIBES BOTH SOLUTIONS IN SOME DETAIL.

MODULARITY EXAMPLE

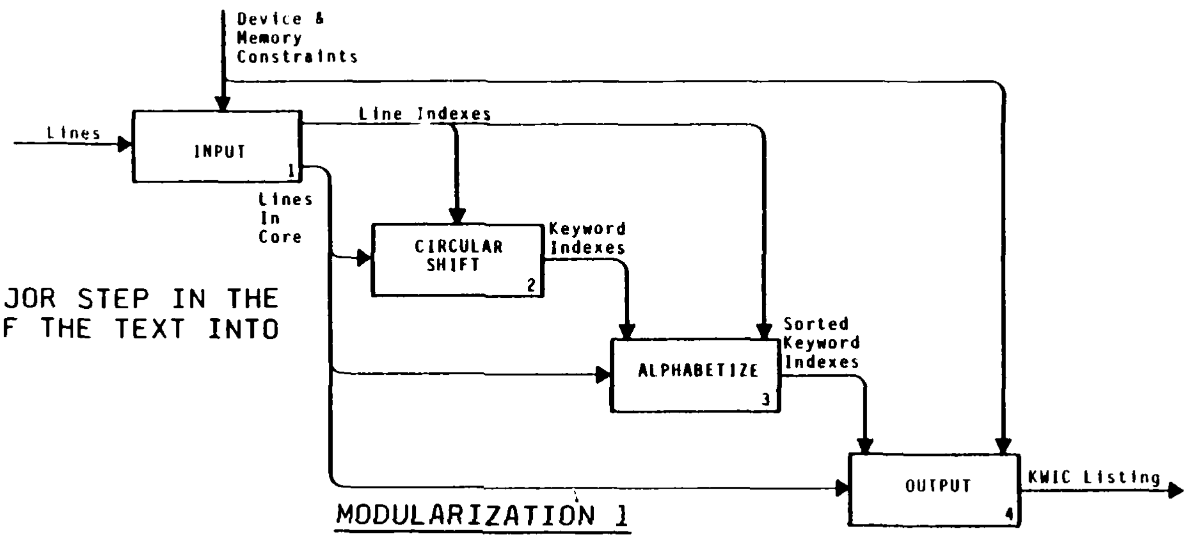
- PROBLEM: KEY WORD IN CONTEXT (KWIC) INDEX SYSTEM
  - IT WILL ACCEPT LINES OF TEXT
  - EACH WORD IN EACH LINE OF TEXT MUST BE MOVED TO THE END, FORMING A NEW LINE
  - EACH LINE (ORIGINAL OR TRANSPOSED) MUST THEN BE PRINTED IN ALPHABETICAL ORDER

THIS FIRST APPROACH IS A TYPICAL SOLUTION. HOWEVER, MUCH DETAILED KNOWLEDGE ABOUT INTERNAL REPRESENTATION IS REQUIRED BY ALL MODULES (I.E., THEY ALL KNOW HOW LINES ARE STORED IN MEMORY AND HOW THE INDEXES ARE MAINTAINED).

IN THE SECOND EXAMPLE, THE DETAIL IS MORE HIDDEN THAN IN THE FIRST ATTEMPT. IN THE SECOND EXAMPLE EACH OF THE MAJOR MODULES (I.E., LINE STORAGE, CIRCULAR SHIFTER, ALPHABETIZE) ARE REPRESENTED BY PROCEDURE CALLS WHICH PROVIDE USING CHAR (r, w, c) AS AN EXAMPLE, THE cTH CHARACTER, OF THE wTH WORD OF THE rTH LINE. SIMILARLY CSCHAR (l, w, c) PROVIDES THE VALUE REPRESENTING THE cTH CHARACTER IN THE wTH WORD OF THE lTH CIRCULAR SHIFT. IN THIS MANNER EACH MODULE MAINTAINS ITS OWN DATA, HIDDEN FROM OTHER MODULES. THEY MAY WELL USE INDICES JUST LIKE THE FIRST MODULARIZATION, AND LINES MAY BE STORED IN MEMORY IN THE SAME MANNER BUT EACH OF THESE "SECRETS" IS ONLY KNOWN BY ONE MODULE AND HIDDEN FROM ALL THE REST.

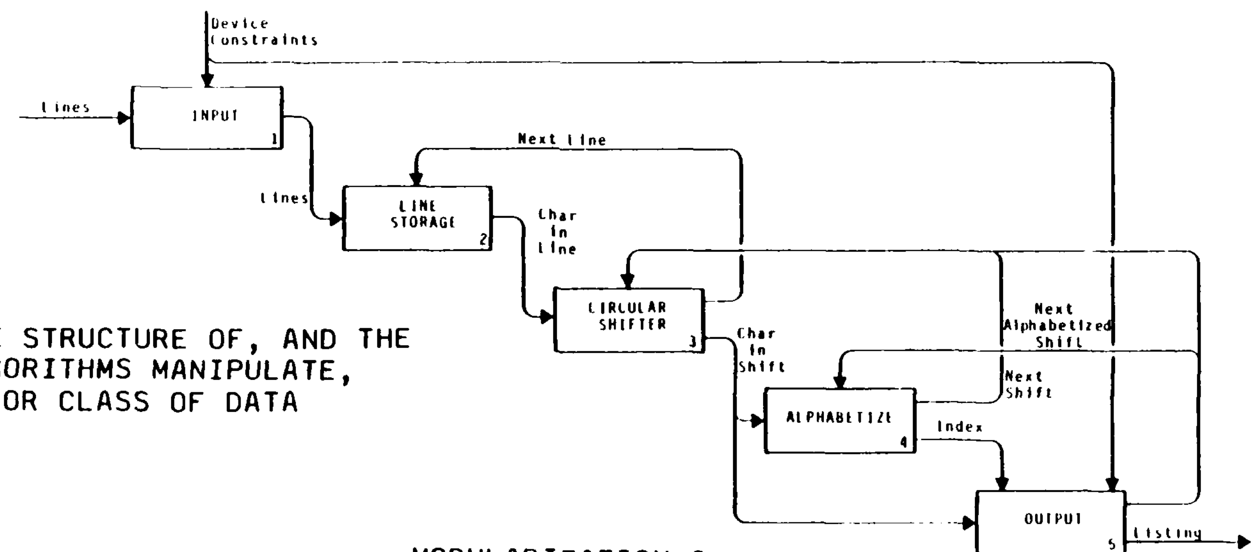
MODULARITY EXAMPLE

- MAKE EACH MAJOR STEP IN THE PROCESSING OF THE TEXT INTO A MODULE



MODULARIZATION 1

- HIDE THE STRUCTURE OF, AND THE WAYS ALGORITHMS MANIPULATE, EACH MAJOR CLASS OF DATA



MODULARIZATION 2

ONE NEEDS TO EXAMINE EACH MODULARIZATION IN THE LIGHT OF ANTICIPATED CHANGES. THERE IS PROBABLY SOME PENALTY THAT WE HAVE PAID FOR THE USE OF CHANGE - MODULARIZATION 2 MAY WELL REQUIRE MORE SUBROUTINE CALLS AND THEREFORE BE SLOWER THAN MODULARIZATION 1.

IF WE LOOK AT MODULARIZATION 2 WE WILL SEE THAT ITS STRUCTURE ANTICIPATED THESE TYPES OF CHANGE - WHICH ARE FAIRLY TYPICAL AND SHOULD HAVE BEEN ANTICIPATED. MODULARIZATION 2 WOULDN'T HELP MUCH AT ALL IF WE CHANGED THE ORDER OF CHARACTERS IN WORDS, OR THE ORDER OF WORDS IN LINES - BUT WE DON'T EXPECT THOSE TYPES OF CHANGES - THEY ARE CONSIDERED INVARIANT.

MODULARITY EXAMPLE

- WHAT HAPPENS WHEN THE DATA EXCEEDS MAIN MEMORY LIMITS AND THE SOFTWARE MUST USE DISK?

  <u>MODULARIZATION 1</u>: CHANGE EVERY MODULE.

  <u>MODULARIZATION 2</u>: CHANGE ONLY THE LINE STORAGE MODULE.

- WHAT HAPPENS IF WE WANT TO DISTRIBUTE ALPHABETIZING OVER THE TIME REQUIRED TO PRINT THE INDEX?

  <u>MODULARIZATION 1</u>: REDESIGN BOTH ALPHABETIZE AND OUTPUT MODULES.

  <u>MODULARIZATION 2:</u> REDESIGN ONLY ALPHABETIZE MODULE.

CONCLUSIONS

- MODULARIZATION 2 IS LESS SENSITIVE TO CHANGES

INSTRUCTOR NOTES

THE MAJOR CHALLENGE IS IN KNOWING WHAT IS LIKELY TO CHANGE. EXPERIENCE WITH SIMILAR SYSTEMS IS THE BEST MEANS OF ATTAINING THIS KNOWLEDGE. VERY OFTEN SOMEONE POSSESSES THIS KNOWLEDGE, BUT IT ISN'T PASSED ON TO THE PERSON MAKING THE MODULARIZATION DECISIONS. ASKING THE ANALYST FOR THIS TYPE OF INFORMATION ABOUT EACH ASPECT OF THE REQUIREMENTS IN A WELL STRUCTURED INTERVIEW IS THE APPROACH USED ON THE SCRP.

ONCE THIS BASIC INFORMATION HAS BEEN GATHERED WE MODULARIZE ON THE BASIS OF HIDING THE IMPACT OF THESE CHANGES - DEVELOPING MODULES WITH INTERFACES TO THEIR USERS THAT WON'T NEED TO BE CHANGED WHEN AN ANTICIPATED CHANGE OCCURS. WE CAN PUT MORE THAN ONE ANTICIPATED CHANGE AREA IN A MODULE, ESPECIALLY IF THEY ARE LIKELY TO CHANGE TOGETHER.

HIDING

- HIDING = KEEPING SECRET HOW ALGORITHMS WORK AND HOW DATA STRUCTURES LOOK

- HIDING MAKES PROGRAM EVOLUTION EASIER

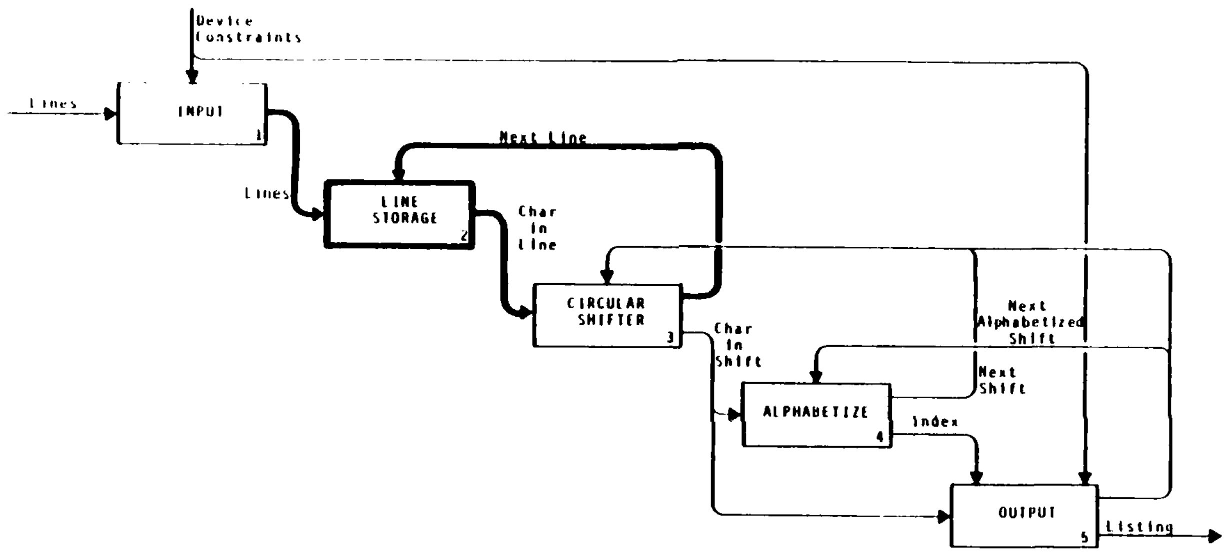
- <u>METHOD</u>:
  - IDENTIFY DESIGN DECISIONS THAT ARE LIKELY TO CHANGE
  - HIDE THOSE DECISIONS INSIDE MODULES
  - DETERMINE THE MODULES' INTERFACE

INSTRUCTOR NOTES

WE KNOW THESE THINGS ARE LIKELY TO CHANGE BECAUSE WE HAVE EXPERIENCE WITH THEM IN THE PAST - SOME CHANGES, LIKE THE REQUIREMENT TO STORE DATA ON DISK INSTEAD OF MEMORY IS A POTENTIAL CHANGE IN ANY SYSTEM - SOMETHING ALL DESIGNS SHOULD ACCOMMODATE.

THE INVARIANT ASSUMPTIONS DESCRIBE THE LOGICAL PROPERTY OF ENGLISH LANGUAGE TEXT - THESE WOULD ONLY CHANGE IF OUR SYSTEM HAD TO DEAL WITH SOMETHING OTHER THAN TEXT - IN THIS CASE WE ARE MAKING THE DECISION THAT IS UNLIKELY.

HIDING EXAMPLE



- <u>DESIGN DECISIONS</u> THAT ARE LIKELY TO CHANGE
  - PHYSICAL STRUCTURE OF A LINE OF TEXT
  - WHERE THE DATA STRUCTURE RESIDES
  - HOW WORDS ARE SEPARATED FROM EACH OTHER

- <u>INVARIANT ASSUMPTIONS</u>
  - TEXT HAS THE FOLLOWING LOGICAL STRUCTURE:

TEXT $\xrightarrow{\text{COMPRISES MANY}}$ LINE $\xrightarrow{\text{COMPRISES MANY}}$ WORD $\xrightarrow{\text{COMPRISES MANY}}$ CHAR

INSTRUCTOR NOTES

FROM THESE PROCEDURE DEFINITIONS YOU CAN SEE HOW THE ACTUAL STORAGE TECHNIQUE - WHETHER IN MEMORY OR ON DISK - WHETHER A LINKED LIST OR ARRAY - IS HIDDEN FROM THE CALLER - ALSO THE USE OF ANY SPECIAL CHARACTER TO DELIMIT WORDS OR LINES IS HIDDEN - ALL OF THESE COULD BE CHANGED WITH NO IMPACT ON THE USERS OF THESE PROCEDURES.

HIDING EXAMPLE

- <u>INTERFACE FOR LINE STORAGE MODULE</u>:

  Store Char (Char, l, w, c)
  Get Char (l, w, c, Char)
  Lines (NumLines)
  Words (l, NumWords)
  Chars (l, w, NumChars)

  HIDES CHANGEABLE DESIGN DECISIONS

- <u>WHERE</u>:

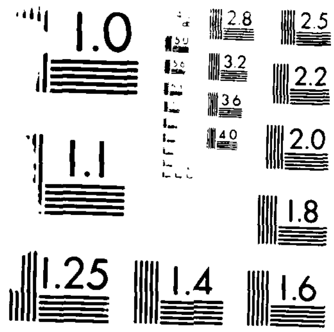
  "Char" IS THE <u>c</u>TH CHARACTER IN THE <u>w</u>TH WORD OF THE <u>l</u>TH LINE,
  "NumLines" IS THE NUMBER OF LINES IN THE TEXT,
  "NumWords" IS THE NUMBER OF WORDS
  IN THE lTH LINE, AND "NumChars" IS THE NUMBER OF CHARACTERS
  IN wTH WORD OF THE lTH LINE.

INSTRUCTOR NOTES

IN SUMMARY THE SCRP MODULARIZATION CRITERIA IS TO REDUCE THE IMPACT OF CHANGE BY ANTICIPATING THE LIKELY CHANGES AND HIDING THE IMPACT OF THESE CHANGES BEHIND INTERFACES THAT CAN BE MAINTAINED REGARDLESS OF THE CHANGE. WE DON'T NEED A SEPARATE MODULE FOR EACH DECISION WE ARE HIDING - JUST HOW MANY DECISIONS TO HIDE IN A MODULE IS DEPENDENT ON THEIR LIKELIHOOD OF CHANGING TOGETHER AND ON OUR ABILITY TO DEFINE INVARIANT INTERFACES FOR THE SINGLE OR MULTIPLE MODULES.

AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2 (U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506 3/6

UNCLASSIFIED F/G 5/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

HIDING

- MODULARIZATION DESERVES CAREFUL THOUGHT:
  - HIDING DESIGN DECISIONS MINIMIZES THE IMPACT OF CHANGING ANY ONE DECISION
  - HIDING HOW A MODULE DOES ITS JOB REDUCES THE INTERDEPENDENCE (COUPLING) BETWEEN MODULES
- SUCCESS DEPENDS ON:
  - ABILITY TO ANTICIPATE CHANGES
  - PROVIDING AN INTERFACE THAT CAN REMAIN INVARIANT

INSTRUCTOR NOTES

SCRP PROJECT HAS BEEN ONGOING FOR MORE THAN FIVE (5) YEARS, MAINLY DUE TO ERRATIC FUNDING. IT IS AN ATTEMPT TO REDEVELOP A COMPLEX, TIME-CRITICAL APPLICATION USING MODERN SOFTWARE ENGINEERING PRINCIPLES. THE INTENT IS TO QUANTIFY THE BENEFITS AND THE COSTS BY MAINTAINING BOTH SYSTEMS IN PARALLEL FOR SEVERAL YEARS WHILE AT THE SAME TIME COLLECTING PERFORMANCE DATA ON BOTH SYSTEMS. THE TRACKING OF MAINTENANCE COSTS HAS ALREADY BEGUN ON THE REQUIREMENTS DESIGN SPECS - SEVERAL PAPERS HAVE BEEN WRITTEN.

THE PHASED INTEGRATION OF THE SOFTWARE HAS BEGUN - THE FIRST BUILD TOOLS A FEW DAYS TO GET FULLY FUNCTIONAL.

SOFTWARE COST REDUCTION PROJECT (SCRP) DESIGN METHOD

- EVOLVED OUT OF A "REAL LIFE" EXPERIMENT TO REDUCE THE COST OF MAINTAINING REALTIME SOFTWARE

- THE EXPERIMENT INCLUDED ...
  - REDEVELOPING AND MAINTAINING THE FLIGHT CONTROL OPERATIONAL PROGRAM FOR THE A-7E AIRCRAFT
  - WORK DONE OUT OF THE NAVAL RESEARCH LAB (NRL)
  - DESIGN FLOWS EASILY FROM SCRP REQUIREMENT SPEC. (SEE SECTION 11)

- METHODS BEING USED BY ...
  - BELL LABS
  - GRUMANN
  - OTHERS

INSTRUCTOR NOTES

SEE 14-20i - THIS IS REALLY PRETTY REDUNDANT - DISCUSS THE FACT THAT THE SCRP HAS PRODUCED PAPERS AND SEVERAL DESIGN SPECIFICATIONS WHICH SHOW MANY WORKED OUT EXAMPLES OF THESE PRINCIPLES. THE VARIOUS TRADE-OFFS THAT WERE CONSIDERED AS WELL AS THE FINAL CONCLUSION ARE WELL DOCUMENTED.

THE ABSTRACT INTERFACE ONLY SHOWS YOU WHAT YOU NEED TO KNOW TO USE A MODULE - JUST AS A ROADMAP ONLY SHOWS YOU WHAT YOU NEED TO KNOW TO DRIVE A CAR - NOT WHERE TREES AND STORES ARE - MAP ALSO HIDE IRRELEVANT DETAILS SO THEY DON'T HAVE TO BE CHANGED WHENEVER SOMETHING THAT DOESN'T EFFECT THE DRIVER IS CHANGED.

SCRP VIEW OF ABSTRACTION, MODULARITY AND HIDING

- PARTITION SYSTEM ON THE BASIS OF EXPECTED CHANGE
  - HIDE EFFECTS OF CHANGE IN SEPARATE MODULES
  - ABSTRACT INTERFACE ONLY SHOWS UNCHANGING ASPECTS OF THE INTERFACE
  - ABSTRACT INTERFACE IS TO MODULE, AS ROAD MAP IS TO WORLD

INSTRUCTOR NOTES

THESE ARE ADDITIONAL TESTS THAT CAN BE APPLIED TO SEE IF A PROPOSED MODULARIZATION IS "GOOD". THESE ARE JUST AN ELABORATION OF PREVIOUSLY STATED OBJECTIVES.

SCRP DESIGN DECOMPOSITION GOALS

- SIMPLE MODULES THAT CAN BE FULLY UNDERSTOOD
- A MODULE IS A WORK ASSIGNMENT FOR A PROGRAMMER OR PROGRAMMER TEAM
- CHANGE TO ONE MODULE'S IMPLEMENTATION SHOULD NOT
  - REQUIRE KNOWLEDGE OF OTHER MODULE'S IMPLEMENTATION
  - AFFECT THE BEHAVIOR OF OTHER MODULES
- EASE OF MAKING A CHANGE SHOULD BE RELATED TO THE LIKELIHOOD OF THE CHANGE BEING NEEDED
- A MAJOR REVISION SHOULD BE ABLE TO BE MADE AS A SET OF INDEPENDENT CHANGES TO INDIVIDUAL MODULES

INSTRUCTOR NOTES

THESE TOP TWO (OF THREE) LEVELS OF THE DESIGN DECOMPOSITION ARE "LOGICAL" OR ORGANIZATIONAL IN NATURE. THE DESIGN DOCUMENTS ARE ORGANIZED THIS WAY - THEY AID THE IMPLEMENTOR OR MAINTAINER IN FINDING WHERE A PARTICULAR FUNCTION HAS BEEN IMPLEMENTED. YOU WON'T NECESSARILY FIND THAT THE CODE GROUPING REFLECTS THE TOP TWO LEVELS - IT REFLECTS ONLY THE LOWEST (THIRD) LEVEL GROUPING.

THE FIRST LEVEL FOR CERTAIN AND PROBABLY THE SECOND AS WELL REPRESENTS A "GENERIC" DESIGN PARTITIONING (I.E., ONE THAT WOULD APPLY TO ALMOST ANY EMBEDDED REAL TIME SYSTEM). DAVE PARNAS HAS BET HIS "RIGHT ARM" ON THE APPLICABILITY OF THE FIRST LEVEL AND THE "BIG TOE OF HIS LEFT FOOT" ON THE SECOND LEVEL. THIS GENERIC ORGANIZATION IS ESSENTIAL TO THE REUSEABILITY OF SOFTWARE IN GENERAL.

SCRP DESIGN DECOMPOSITION

(GENERIC MODULE HIERARCHY GUIDELINES)

- HIERARCHY OF MODULES ALLOW RELEVANT MODULES TO BE IDENTIFIED BY THE READER
- GENERIC HIERARCHY PROVIDES A STARTING POINT FOR "ALL" SOFTWARE IMPLEMENTATIONS
- FIRST LEVEL HIERARCHY / SECOND LEVEL HIERARCHY

FIRST LEVEL HIERARCHY	SECOND LEVEL HIERARCHY
HARDWARE HIDING MODULE	EXTENDED COMPUTER MODULE
	DEVICE INTERFACE MODULE
BEHAVIOR HIDING MODULE	FUNCTION DRIVER MODULE
	SHARED SERVICES MODULE
SOFTWARE DECISION MODULE	APPLICATION DATA MODULE
	PHYSICAL MODEL MODULE
	DATA BANKER MODULE
	SYSTEM GENERATION MODULE
	SOFTWARE UTILITY MODULE

INSTRUCTOR NOTES

THE ROLE A MODULE PLAYS IS ITS FUNCTIONALITY - THE PORTION OF THE REQUIREMENTS THAT IT IMPLEMENTS. THE SECRETS ARE THE DESIGN DECISIONS THAT ARE HIDDEN IN EACH MODULE. THE FACILITIES ARE THE FUNCTION CALLS, PROCEDURE CALLS, VARIABLES THAT A MODULE MAKES VARIABLE TO OTHER MODULES IN THE SYSTEM (I.E., ITS ABSTRACT INTERFACE).

IN DESCRIBING THE TOP LEVEL OF THE GENERIC HIERARCHY WE DISCUSS THE FIRST TWO OF THESE - THE THIRD IS VERY DETAILED AND HARD TO SUMMARIZE.

THE HARDWARE HIDING MODULE MAKES IT POSSIBLE TO TRANSPORT ALL OF THE REST OF THE SYSTEM TO ANOTHER COMPUTER. ONLY THE HARDWARE HIDING MODULE "KNOWS" ABOUT THE IDIOSYNCRISES OF THIS COMPUTER. IT INCLUDES THE I/O INTERFACE SECRETS.

THE THREE LEVELS OF THE SOFTWARE DESIGN HIERARCHY ARE DESCRIBED IN THE A-7E SOFTWARE MODULE GUIDE **KEEP STRESSING THE GENERIC APPLICABILITY OF THE TOP TWO LEVELS OF THE HIERARCHY**.

GENERIC MODULE HIERARCHY (CONTINUED)

- THREE WAYS TO DESCRIBE A MODULE STRUCTURE BY
  - THE ROLE INDIVIDUAL MODULES PLAY IN SYSTEM OPERATION
  - THE SECRETS ASSOCIATED WITH EACH MODULE
  - THE FACILITIES PROVIDED BY EACH MODULE

- HARDWARE HIDING MODULE INCLUDES
  - PROGRAMS NEEDED TO CHANGE IF ANY HARDWARE OF THE SYSTEM CHANGES BUT WHICH STILL HAS THE SAME GENERAL CAPABILITIES
  - IMPLEMENTS "VIRTUAL HARDWARE"
  - HIDES "SECRETS" OF SECTION 1 AND 2 OF REQUIREMENTS SPEC. (I.E. DATA STRUCTURES AND ALGORITHMS)

INSTRUCTOR NOTES

THE BEHAVIOR HIDING MODULE HIDES THE DETAILS OF ALL THE CONDITION AND EVENT TABLES FROM SECTIONS 3 AND 4 OF THE REQUIREMENTS SPECIFICATIONS. IT ALSO HIDES JUST HOW MODES ARE IMPLEMENTED OR DETERMINED.

THE SOFTWARE DECISION MODULE SOLELY CONTAINS DESIGN DECISIONS THAT ARE MOTIVATED BY IMPLEMENTATION CONSIDERATIONS SUCH AS EFFICIENCY AND ACCURACY. NOTE THAT SOME ALGORITHMS AND SOME ACCURACY CONSTRAINTS ARE DEFINED IN THE REQUIREMENTS SPEC. IN THESE CASES THESE DESIGN DECISIONS ARE NOT PART OF THIS MODULE - THEY WOULD PROBABLY BE A PART OF THE BEHAVIOR HIDING MODULE.

GENERIC MODULE HIERARCHY (CONTINUED)

- BEHAVIOR-HIDING MODULE INCLUDES
  - PROGRAMS THAT WOULD CHANGE IN SECTION 3 OR 4 OF REQUIREMENT SPECIFICATION CHANGED
  - THESE PROGRAMS DETERMINE THE VALUES SENT TO THE VIRTUAL OUTPUTS PROVIDED BY HARDWARE-HIDING MODULE
- SOFTWARE DECISION MODULE INCLUDES
  - HIDES SOFTWARE DECISIONS THAT ARE BASED ON
    - MATHEMATICAL THEREOMS
    - PHYSICAL FACTS
    - ALGORITHMIC EFFICIENCY
    - ALGORITHMIC ACCURACY
  - NO DIRECT RELATIONSHIP TO REQUIREMENT SPECIFICATION
  - THE "SECRETS" IN THIS MODULE ARE <u>NOT</u> DESCRIBED IN REQUIREMENT DOCUMENT

THE EXTENDED COMPUTER MODULE HIDES THOSE CHARACTERISTICS OF HARDWARE/SOFTWARE INTERFACE OF THE AVIONICS COMPUTER THAT WE CONSIDER LIKELY TO CHANGE IF THE COMPUTER IS MODIFIED OR REPLACED.

THE PRIMARY SECRETS OF THE EXTENDED COMPUTER ARE: THE NUMBER OF PROCESSORS, THE INSTRUCTION SET OF THE COMPUTER, AND THE COMPUTER'S CAPACITY FOR PERFORMING CONCURRENT OPERATIONS.

THE DEVICE INTERFACE MODULE HIDES THE CHARACTERISTICS OF THE PERIPHERAL DEVICES THAT ARE CONSIDERED LIKELY TO CHANGE. EACH DEVICE MIGHT BE REPLACED BY AN IMPROVED DEVICE CAPABLE OF ACCOMPLISHING THE SAME TASKS. REPLACEMENT DEVICES DIFFER WIDELY IN THEIR HARDWARE/SOFTWARE INTERFACES. FOR EXAMPLE, ALL ANGLE-OF-ATTACK SENSORS MEASURE ANGLE-OF-ATTACK, BUT THEY DIFFER IN INPUT FORMAT, TIMING, AND THE AMOUNT OF NOISE IN THE DATA. THE DEVICE INTERFACE MODULE PROVIDES VIRTUAL DEVICES TO BE USED BY THE REST OF THE SOFTWARE. THE VIRTUAL DEVICES DO NOT NECESSARILY CORRESPOND TO PHYSICAL DEVICES BECAUSE ALL OF THE HARDWARE PROVIDING A CAPABILITY IS NOT NECESSARILY IN ONE PHYSICAL UNIT. FURTHER, THERE ARE SOME CAPABILITIES OF A PHYSICAL UNIT THAT ARE LIKELY TO CHANGE INDEPENDENTLY OF OTHERS: IT IS ADVANTAGEOUS TO HIDE CHARACTERISTICS THAT MAY CHANGE INDEPENDENTLY IN DIFFERENT MODULES.

THE PRIMARY SECRETS OF THE DEVICE INTERFACE MODULE ARE THOSE CHARACTERISTICS OF THE PRESENT DEVICES DOCUMENTED IN THE REQUIREMENTS DOCUMENT AND NOT LIKELY TO BE SHARED BY REPLACEMENT DEVICES.

OUR DISTINCTION BETWEEN COMPUTER AND DEVICE IS BASED ON THE CURRENT HARDWARE AND IS THE ONE MADE IN THE REQUIREMENTS DOCUMENT. INFORMATION THAT APPLIES TO MORE THAN ONE DEVICE IS CONSIDERED A SECRET OF THE EXTENDED COMPUTER; INFORMATION THAT IS ONLY RELEVANT TO ONE DEVICE IS A SECRET OF A DEVICE INTERFACE MODULE.

THE FUNCTION DRIVER MODULE CONSISTS OF A SET OF INDIVIDUAL MODULES CALLED FUNCTION DRIVERS; EACH FUNCTION DRIVER IS THE SOLE CONTROLLER OF A SET OF CLOSELY RELATED OUTPUTS. THE OUTPUTS ARE EITHER PART OF A VIRTUAL DEVICE OR PROVIDED BY THE EXTENDED COMPUTER FOR TEST PURPOSES. THE PRIMARY SECRETS OF THE FUNCTION DRIVER MODULE ARE THE RULES DETERMINING THE VALUES OF THESE OUTPUTS.

SECOND LEVEL HIERARCHY GUIDELINES

- EXTENDED COMPUTER MODULE
  - HIDES THE NUMBER OF PROCESSORS, THE INSTRUCTION SET, CAPACITY FOR CONCURRENT OPERATION, MEMORY MANAGEMENT
  - HIDES COMPUTER CHARACTERISTICS LIKELY TO CHANGE (E.G. FLOATING POINT SUPPORT)
- DEVICE *INTERFACE MODULE*
  - HIDES CHARACTERISTICS OF DEVICE NOT LIKELY TO BE SHARED WITH REPLACEMENT DEVICES
  - PROVIDES "VIRTUAL DEVICES" TO OTHER SOFTWARE
- FUNCTION DRIVER MODULE
  - CONSISTS OF A SET OF MODULES EACH OF WHICH IS THE SOLE CONTROLLER OF A SET OF CLOSELY RELATED OUTPUTS
  - HIDES THE RULES FOR DETERMINING THE VALUES OF OUTPUTS

BECAUSE ALL THE FUNCTION DRIVERS CONTROL SYSTEMS IN THE SAME AIRCRAFT, SOME ASPECTS OF THE BEHAVIOR ARE COMMON TO SEVERAL FUNCTION DRIVERS. WE EXPECT THAT IF THERE IS A CHANGE IN THAT ASPECT OF THE BEHAVIOR, IT WILL AFFECT ALL OF THE FUNCTIONS THAT SHARE IT. CONSEQUENTLY WE HAVE IDENTIFIED A SET OF MODULES, EACH OF WHICH HIDES AN ASPECT OF THE BEHAVIOR THAT APPLIES TO TWO OR MORE OF THE OUTPUTS.

BECAUSE USERS OF THE DOCUMENTATION CANNOT BE EXPECTED TO KNOW WHICH ASPECTS OF A FUNCTION'S BEHAVIOR ARE SHARED, THE DOCUMENTATION FOR THE FUNCTION DRIVER MODULES WILL INCLUDE A REFERENCE TO THE SHARED SERVICES MODULES THAT IT USES. A MAINTENANCE PROGRAMMER SHOULD ALWAYS BEGIN HIS INQUIRY WITH THE APPROPRIATE FUNCTION DRIVER. HE WILL BE DIRECTED TO THE SHARED SERVICES MODULES WHEN APPROPRIATE.

THE APPLICATION DATA TYPE MODULE SUPPLEMENTS THE DATA TYPES PROVIDED BY THE EXTENDED COMPUTER MODULE WITH DATA TYPES THAT ARE PARTICULARLY USEFUL FOR AVIONICS APPLICATIONS AND DO NOT REQUIRE A COMPUTER DEPENDENT IMPLEMENTATION. THESE DATA TYPES ARE IMPLEMENTED USING THE DATA TYPES PROVIDED BY THE EXTENDED COMPUTER; VARIABLES OF THOSE TYPES ARE USED JUST AS IF THE TYPES WERE BUILT INTO THE EXTENDED COMPUTER.

THE SECRETS OF THE APPLICATION DATA TYPE MODULE ARE THE DATA REPRESENTATION USED IN THE VARIABLES AND THE PROGRAMS USED TO IMPLEMENT OPERATIONS ON THOSE VARIABLES. THESE VARIABLES CAN BE USED WITHOUT CONSIDERATION OF UNITS. WHERE NECESSARY, THE MODULES PROVIDE CONVERSION OPERATORS, WHICH DELIVER OR ACCEPT REAL VALUES IN SPECIFIED UNITS.

THE SOFTWARE REQUIRES ESTIMATES OF QUANTITIES THAT CANNOT BE MEASURED DIRECTLY BUT CAN BE COMPUTED FROM OBSERVABLES USING MODELS OF THE PHYSICAL WORLD. THE PRIMARY SECRETS OF THE PHYSICAL MODEL MODULE ARE THE PHYSICAL MODELS; THE SECONDARY SECRETS ARE THE COMPUTER IMPLEMENTATION OF THOSE MODELS.

SECOND LEVEL HIERARCHY GUIDELINES (CONTINUED)

- SHARED SERVICES MODULE
  - CONTAINS THE PROGRAM SEGMENTS THAT ARE COMMON TO FUNCTION DRIVERS
  - MAY BE SUB-FUNCTIONS OF A FUNCTION DRIVER

- APPLICATION DATA MODULE
  - PROVIDES IMPLEMENTATION OF APPLICATION SPECIFIC DATA TYPES NOT DIRECTLY PROVIDED BY THE EXTENDED COMPUTER MODULE
  - HIDES DATA REPRESENTATION USED AND ALLOWED OPERATIONS ON VARIABLES WITH THAT REPRESENTATION

- PHYSICAL MODEL MODULE
  - HIDES ANY IMPLEMENTATION OF PHYSICAL MODELS NEEDED TO COMPUTE QUANTITIES THAT CAN NOT BE DIRECTLY MEASURED

MOST DATA ARE PRODUCED BY ONE MODULE AND CONSUMED BY ANOTHER. IN MANY CASES, THE CONSUMERS SHOULD RECEIVE A VALUE AS UP-TO-DATE AS PRACTICAL. THE DATA BANK MODULE ACTS AS A "MIDDLEMAN" AND DETERMINES WHEN NEW VALUES FOR THESE DATA ARE COMPUTED. THE DATA BANKER OBTAINS VALUES FROM PRODUCERS; CONSUMER PROGRAMS OBTAIN DATA FROM DATA BANKER ACCESS PROGRAMS. THE PRODUCER AND CONSUMERS OF A PARTICULAR DATUM CAN BE WRITTEN WITHOUT KNOWING WHETHER OR NOT THE DATA BANKER STORES THE VALUE OR WHEN A STORED VALUE IS UPDATED. IN MOST CASES, NEITHER THE PRODUCER NOR THE CONSUMER NEED BE MODIFIED IF UPDATING POLICY CHANGES.

THE DATA BANKER IS USED REGARDLESS OF THE FREQUENCY OF THE UPDATES; FOR EXAMPLE, IT PROVIDES VALUES OF DATA RECEIVED THROUGH THE PANEL, EVEN THOUGH THEY ARE RARELY UPDATED. THE DATA BANKER IS USED AS LONG AS CONSUMER AND PRODUCER ARE SEPARATE MODULES, EVEN WHEN THEY ARE BOTH SUBMODULES OF A LARGER MODULE.

THE PRIMARY SECRETS OF THE SYSTEM GENERATION MODULE ARE DECISIONS THAT ARE POSTPONED UNTIL SYSTEM-GENERATION TIME. THESE INCLUDE THE VALUES OF SYSTEM GENERATION PARAMETERS AND THE CHOICE AMONG ALTERNATIVE IMPLEMENTATIONS OF A MODULE. THE SECONDARY SECRETS OF THE SYSTEM GENERATION MODULE ARE THE METHOD USED TO GENERATE A MACHINE-EXECUTABLE FORM OF THE CODE AND THE REPRESENTATION OF THE POSTPONED DECISIONS. MOST OF THE PROGRAMS IN THIS MODULE DO NOT RUN ON THE ON-BOARD COMPUTER; THEY RUN ON THE COMPUTER USED TO GENERATE THE CODE FOR THE ON-BOARD SYSTEM. NOTE THAT THESE DECISIONS, ALTHOUGH VERY IMPORTANT ARE OFTEN NOT CONSIDERED TO BE PART OF THE SYSTEM. THEY ARE DOCUMENTED AND IMPLEMENTED IN AN ADHOC MANNER.

<u>SOFTWARE UTILITY MODULE</u> - THE PRIMARY SECRETS OF THIS MODULE ARE THE ALGORITHMS <u>IMPLEMENTING COMMON SOF</u>TWARE FUNCTIONS SUCH AS START PROCESS, AND MATHEMATICAL FUNCTIONS SUCH AS SQUARE-ROOT AND LOGARITHM.

<u>RESOURCE MONITOR MODULE</u> - THE PRIMARY SECRETS OF THIS MODULE ARE THE SYNCHRONIZATION <u>ALGORITHMS USED TO PROV</u>IDE ORDERLY ACCESS TO SHARED RESOURCES SUCH AS DEVICES AND DATA STRUCTURES. PROGRAMS FROM THESE MODULES ARE USED WITHIN MODULES THAT PROVIDE VIRTUAL RESOURCES.

SECOND LEVEL HIERARCHY GUIDELINES (CONTINUED)

- DATA BANKER MODULE
  - HIDES THE DATABASE FOR THE APPLICATION
  - IMPLEMENTS ACCESS AND DATA MAINTENANCE POLICIES
- SYSTEM GENERATION MODULE
  - HIDE DECISIONS THAT ARE POSTPONED UNTIL SYSTEM GENERATION TIME
  - MAY INCLUDE SYSTEM GENERATION PROCEDURAL SOFTWARE (TYPICALLY RUNS ON A HOST MACHINE)
- SOFTWARE UTILITY MODULE
  - IMPLEMENTS COMMON SOFTWARE FUNCTIONS SUCH AS START PROCESS, MATH ROUTINES, ETC.
- RESOURCE MONITOR MODULE
  - IMPLEMENTS (AND HIDES) SYNCHRONIZATION ALGORITHMS

<u>INTERFACE OVERVIEW</u> - A TABLE OF PROGRAMS ON THE MODULE'S INTERFACE, SHOWING THE PARAMETERS AND PARAMETER TYPES AND STATING THE EFFECTS OF EACH.

<u>LOCAL TYPE DEFINITION</u> - DEFINITIONS OF THE DATA TYPES AVAILABLE TO USERS FROM THE MODULE.

<u>SYSTEM GENERATION PARAMETERS</u> - A LIST OF THOSE QUANTITATIVE CHARACTERISTICS OF THE MODULE THAT ARE NOT BOUND UNTIL JUST BEFORE RUN-TIME (E.G., THE SIZE OF A DATA STRUCTURE).

<u>DESIGN ISSUES</u> - A PROSE SECTION EXPLAINING WHY CERTAIN DESIGN DECISIONS WERE MADE TO AID PEOPLE WHO MIGHT MAKE FUTURE CHANGES TO THE DESIGN;

<u>IMPLEMENTATION NOTES</u> - A PROSE SECTION TO CAPTURE INFORMATION THAT MIGHT HAVE COME TO THE DESIGNER'S ATTENTION THAT WOULD BE OF USE TO THE IMPLEMENTORS.

<u>ASSUMPTIONS LISTS</u> - A PROSE SECTION DOCUMENTING THE ASSUMPTIONS THAT THE USERS OF THE MODULE ARE ALLOWED TO MAKE ABOUT IT.

SEVERAL OTHER SECTIONS ARE ALSO INCLUDED, BUT THESE ARE THE MAJOR ONES. THESE SPECIFICATIONS USE A LOT OF SPECIAL SYMBOLS AND TERMINOLOGY TO KEEP THEM AS BRIEF AS POSSIBLE WITH AS MUCH PRECISENESS AS POSSIBLE.

PRIMARY SCRP DESIGN METHOD DOCUMENTATION

- MODULE SPECS (ALSO CALLED ABSTRACT INTERFACE SPECS) ARE USED AS DESIGN REPRESENTATION TOOLS. THEY MUST DESCRIBE:
    - INTERFACE TO THE MODULE
        1) ACCESS PROGRAMS
        2) EVENTS
        3) EFFECTS
    - LOCAL DATA TYPES
    - SYSTEM GENERATION PARAMETERS
    - DESIGN ISSUES
    - IMPLEMENTATION NOTES
    - ASSUMPTIONS

- MODULE SPECS ARE PRECISE AND CONCISE DESCRIPTIONS OF VIABLE BEHAVIOR OF A MODULE

INSTRUCTOR NOTES

THE OBJECTIVE OF THE SCRP WAS TO TRANSITION MODERN SOFTWARE ENGINEERING CONCEPTS AND TECHNIQUES TO PRACTICAL AND WIDESPREAD USE BY PROVIDING A COMPLETELY WORKED OUT EXAMPLE WHICH COULD BE USED AS A MODEL BY OTHER PROJECTS. THE ACTUAL SCRP DOCUMENTS PROVIDE MANY WELL DOCUMENTED EXAMPLES OF FUZZY CONCEPTS SUCH AS HIDING, ABSTRACTION, AND MODULARITY. VARIOUS TRADE-OFFS AND DESIGN EVOLUTIONS ARE WELL DOCUMENTED.

IN ADDITION THE DESIGN STRUCTURE ITSELF IS MEANT TO BE REUSEABLE (AT LEAST THE FIRST TWO LEVELS) AND PROVIDES A GOOD STARTING POINT.

THE CONSISTENT USE OF SEPARATION OF CONCERNS AND HIGH DEGREE OF TRACEABILITY FROM REQUIREMENTS SPECS TO DESIGN SPECS DIRECTLY SUPPORTS REUSEABLE SOFTWARE SINCE WE ARE ABLE TO REUSE APPROPRIATE PARTS OF EACH DOCUMENT AS WELL AS THE CODE FOR THE MODULE ITSELF.

SCRP WRAP-UP

- SCRP DESIGN METHODS PROVIDE ...
  - DIRECT SUPPORT OF ABSTRACTION, MODULARITY AND HIDING CONCEPTS
  - PATH FROM REQUIREMENTS SPEC. TO MODULE DESIGN
  - CRITERIA AND GUIDELINES FOR MODULARIZATION OF REAL TIME SOFTWARE
  - FRAMEWORK FOR DOCUMENTING DESIGNS
  - FRAMEWORK FOR REUSEABLE SOFTWARE COMPONENTS

INSTRUCTOR NOTES

THEME: OBJECT-ORIENTED DESIGN PROVIDES A STRATEGY FOR MODULARIZING AND GETTING STARTED WITH A DESIGN.

PURPOSE: PROVIDE AN OVERVIEW OF A FAIRLY MODERN DESIGN STRATEGY BY WALKING THRU AN EXAMPLE.

REFERENCES: BOOCH, G. "SOFTWARE ENGINEERING WITH Ada" BENJAMIN/CUMMINGS, MA; 1983.

Section 15

OBJECT-ORIENTED DESIGN

INSTRUCTOR NOTES

THIS TECHNIQUE WAS FORMALIZED BY GRADY BOOCH IN HIS BOOK "SOFTWARE ENGINEERING IN Ada." THERE ARE SEVERAL PAPERS BY HIM AND OTHERS ON THIS TOPIC.

ABSTRACT DATA TYPES - SET OF VALUES, DATA STRUCTURES AND THE ASSOCIATED OPERATION RELATED TO THAT DATA TYPE.

OBJECT-ORIENTED DESIGN OVERVIEW

- EVOLVED FROM WORK DONE BY G. BOOCH AND INTEL CORPORATION ON METHODS TO EXPRESS Ada SOFTWARE MODULE DESIGNS

- BASED ON THE SAME CONCEPTS AS SCRP DESIGN METHODS
  - ABSTRACTION
  - MODULARITY
  - HIDING

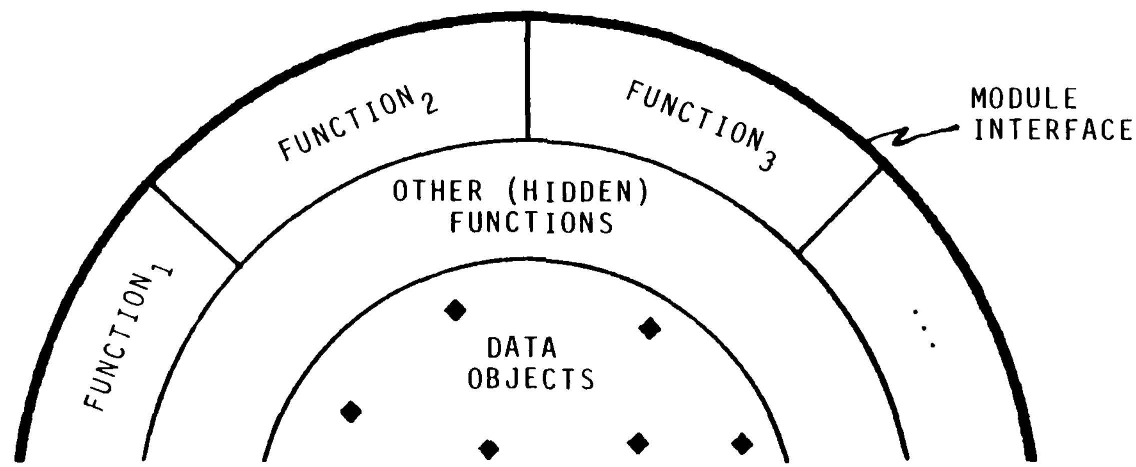
- DIFFERS FROM SCRP DESIGN METHOD IN ...
  - EMPHASIS ON ABSTRACTION OF DATA (ABSTRACT DATA TYPES)
  - CRITERIA FOR MODULARIZATION BASED ON DATA TYPE AND OPERATIONS
  - BOTTOM-UP VS. TOP-DOWN VIEW OF DESIGN

INSTRUCTOR NOTES

"MODULES" ARE COLLECTIONS (PACKAGES) OF Ada PROCEDURES OR FUNCTIONS THAT (BY SOMETIMES CALLING OTHER (HIDDEN) FUNCTIONS) OPERATE ON A SET OF CLOSELY RELATED DATA OBJECTS. FUNCTIONS USUALLY EXIST FOR CREATING, DESTROYING, EXAMINING, AND MANIPULATING DATA OBJECTS.

OBJECT-ORIENTED DESIGN

- "Ada" OBJECT-ORIENTED DESIGN BUILDS A SYSTEM OF MODULES, EACH OPERATING ON A <u>CLASS</u> OF DATA OBJECTS:



- A CLASS OF DATA OBJECTS WOULD BE CHARACTERIZED AS AN ABSTRACT DATA TYPE
  - A SET OF VALUES
  - A DATA STRUCTURE
  - THE ALLOWED OPERATION ON THE DATA STRUCTURE AND SET OF VALUES

INSTRUCTOR NOTES

ONE NEEDS TO IDENTIFY A VERBAL DESCRIPTION OF THE PROBLEM.

DATA OBJECTS - TYPICALLY NOUNS IN THE PROBLEM STATEMENT.

OPERATIONS ON OBJECTS - TYPICALLY VERBS IN THE PROBLEM STATEMENT.

ESTABLISH INTERFACES - DEFINE IN A FORMAL SENSE THE INPUTS/OUTPUTS OF EACH OPERATION.

OBJECT-ORIENTED DESIGN

- <u>METHOD</u>
  - DEVELOP AN INFORMAL STRATEGY
  - IDENTIFY DATA OBJECTS
  - IDENTIFY OPERATIONS ON THOSE OBJECTS
  - ESTABLISH INTERFACES
    - PACKAGE INTO "MODULES"
    - EXPRESS IN TERMS OF A PROGRAM DESIGN LANGUAGE (Ada)

INSTRUCTOR NOTES

TAKE THE CLASS THROUGH A SIMPLE EXAMPLE OF OBJECT-ORIENTED DESIGN IN ACTION.

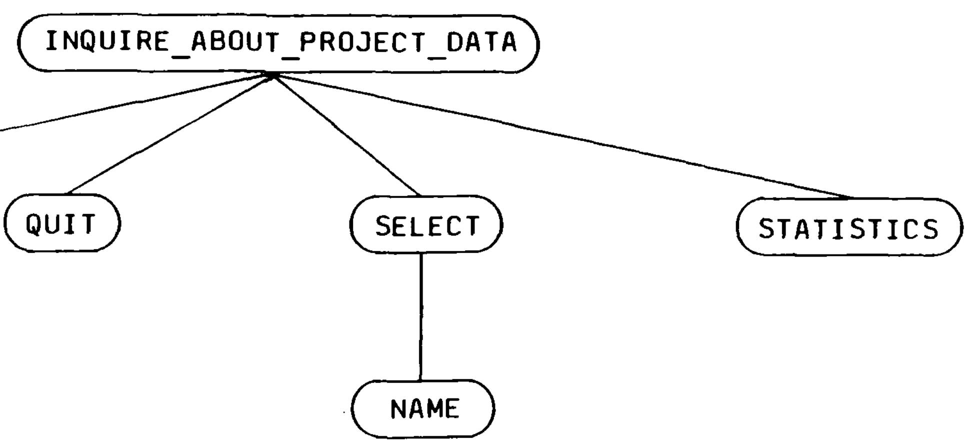
THE TREE STRUCTURE SHOWS AN EXAMPLE COMMAND HIERARCHY.

OBJECT-ORIENTED DESIGN EXAMPLE

(A DATABASE INQUIRY SYSTEM)

- INFORMAL STRATEGY (I.E. PROBLEM STATEMENT)

A DATABASE IN THE APSE EXISTS TO CAPTURE THE INFORMATION FOR EACH PROGRAM UNIT OF A GIVEN PROJECT. THE INFORMATION MAINTAINED ABOUT EACH UNIT INCLUDES THE UNIT NAME, ASSIGNED PROGRAMMER, STATUS OF DEVELOPMENT, CURRENT VERSION, AND REFERENCE TO THE CORRESPONDING REQUIREMENTS DOCUMENT. THE POSSIBLE INTERACTIVE OPERATIONS THAT A USER MAY REQUEST INCLUDE LISTING PROJECT DATA (SORTED BY PROGRAMMER NAME OR PRIMARY REQUIREMENTS), DISPLAYING ALL INFORMATION ON A SELECTED PROGRAM UNIT, AND COLLECTING STATISTICS ON THE LEVEL OF DEVELOPMENT FOR ALL PROGRAM UNITS. AN INQUIRY SESSION WILL CONTINUE UNTIL A USER ENTERS A REQUEST TO QUIT.



EXAMPLE SOURCE: SOFTWARE ENGINEERING WITH Ada, 1983

INSTRUCTOR NOTES

RELATE THE NOUNS TO THE <u>OBJECTS</u> OF INTEREST.

OBJECT-ORIENTED DESIGN EXAMPLE

- IDENTIFY THE DATA OBJECTS (I.E. THE NOUNS IN THE INFORMAL STRATEGY)

A DATABASE IN THE APSE EXISTS TO CAPTURE THE INFORMATION FOR EACH PROGRAM UNIT OF A GIVEN PROJECT. THE INFORMATION MAINTAINED ABOUT EACH UNIT INCLUDES THE UNIT NAME, ASSIGNED PROGRAMMER, STATUS OF DEVELOPMENT, CURRENT VERSION, AND REFERENCE TO THE CORRESPONDING REQUIREMENTS DOCUMENT. THE POSSIBLE INTERACTIVE OPERATIONS THAT A USER MAY REQUEST INCLUDE LISTING PROJECT DATA (SORTED BY PROGRAMMER NAME OR PRIMARY REQUIREMENTS), DISPLAYING ALL INFORMATION ON A SELECTED PROGRAM UNIT, AND COLLECTING STATISTICS ON THE LEVEL OF DEVELOPMENT FOR ALL PROGRAM UNITS. AN INQUIRY SESSION WILL CONTINUE UNTIL A USER ENTERS A REQUEST TO QUIT.

OBJECTS OF INTEREST

- INQUIRY_OPERATIONS
    - COMMAND
- PROJECT
    - UNIT_INFORMATION — DATA_BASE
        - UNIT_NAME
        - PROGRAMMER
        - STATUS
        - VERSION
        - REQUIREMENTS

INSTRUCTOR NOTES

RELATE THE VERBS TO THE <u>OPERATIONS</u> OF INTEREST.

OBJECT-ORIENTED DESIGN EXAMPLE

- IDENTIFY OPERATIONS ON DATA OBJECTS (I.E. VERBS IN INFORMAL STRATEGY)

A DATABASE IN THE APSE EXISTS TO CAPTURE THE INFORMATION FOR EACH PROGRAM UNIT OF A GIVEN PROJECT. THE INFORMATION MAINTAINED ABOUT EACH UNIT INCLUDES THE UNIT NAME, ASSIGNED PROGRAMMER, STATUS OF DEVELOPMENT, CURRENT VERSION, AND REFERENCE TO THE CORRESPONDING REQUIREMENTS DOCUMENT. THE POSSIBLE INTERACTIVE OPERATIONS THAT A USER MAY <u>REQUEST</u> INCLUDE <u>LISTING PROJECT DATA</u> (<u>SORTED</u> BY PROGRAMMER NAME OR PRIMARY REQUIREMENTS), <u>DISPLAYING</u> ALL INFORMATION ON A SELECTED PROGRAM UNIT, AND <u>COLLECTING STATISTICS</u> ON THE LEVEL OF DEVELOPMENT FOR ALL PROGRAM UNITS. AN INQUIRY SESSION WILL CONTINUE UNTIL A USER <u>ENTERS</u> A REQUEST <u>TO QUIT</u>.

<u>OPERATIONS OF INTEREST</u>

- INQUIRY_OPERATIONS
  - COMMAND
    - REQUEST
- PROJECT
  - UNIT_INFORMATION — DATA_BASE
    - COLLECT_STATISTICS
    - LIST_DATA
    - QUIT
    - SELECT_UNIT

INSTRUCTOR NOTES

THE CLASS MAY NOT HAVE DIRECT Ada EXPERIENCE, SO GIVE THEM AN EXPLANATION OF TYPES, PACKAGES, OBJECTS, ETC.

IF YOUR NOT FAMILIAR WITH Ada GO TO ONE OF THE Ada INTRODUCTORY COURSES AND LOOK-UP THESE ITEMS.

PDL = PROGRAM DESIGN LANGUAGE.

OBJECT-ORIENTED DESIGN EXAMPLE

- ESTABLISH INTERFACES USING AN ADA PDL

```
package PROJECT is
   package UNIT_INFORMATION is
      type NAME_TYPE          is new STRING(1 .. 20);
      type PROGRAMMER_TYPE    is new STRING(1 .. 20);
      type REFERENCE;
      type REFERENCE_ACCESS is access REFERENCE;
      type REFERENCE          is record
                                    DOCUMENT : STRING(1 .. 80);
                                    PAGE     : POSITIVE;
                                    NEXT     : REFERENCE_ACCESS;
                                 end record;
      type STATUS_TYPE        is (DESIGN, CODE, TEST, OPERATIONAL);
      type VERSION_TYPE       is range 0 .. 99;
      type UNIT_RECORD        is record
                                    PROGRAMMER   : PROGRAMMER_TYPE;
                                    REQUIREMENTS : REFERENCE_ACCESS;
                                    STATUS       : STATUS_TYPE;
                                    UNIT_NAME    : NAME_TYPE;
                                    VERSION      : VERSION_TYPE;
                                 end record;
   end UNIT_INFORMATION;

   package DATA_BASE is
      use UNIT_INFORMATION;
      MAXIMUM_RECORDS       : constant := 100;
      type RECORD_INDEX     is RANGE 1 .. MAXIMUM_RECORDS;
      type PROJECT_RECORDS is array (RECORD_INDEX) of UNIT_RECORD;
      ACTIVE_RECORDS        : RECORD_INDEX;
      DATA                  : PROJECT_RECORDS;
   end DATA_BASE;
end PROJECT;
```

INSTRUCTOR NOTES

RELATE THE GRAPHIC AND PDL DESCRIPTION.

DESCRIBE WHAT A FUNCTION AND PROCEDURE ARE.

INQUIRY_OPERATIONS IMPLEMENTS AN ABSTRACT DATA TYPE FOR COMMAND

<u>COMMAND SET OF VALUES</u>

- COLLECT_STATISTICS
- LIST_DATA
- QUIT
- SELECT_UNIT

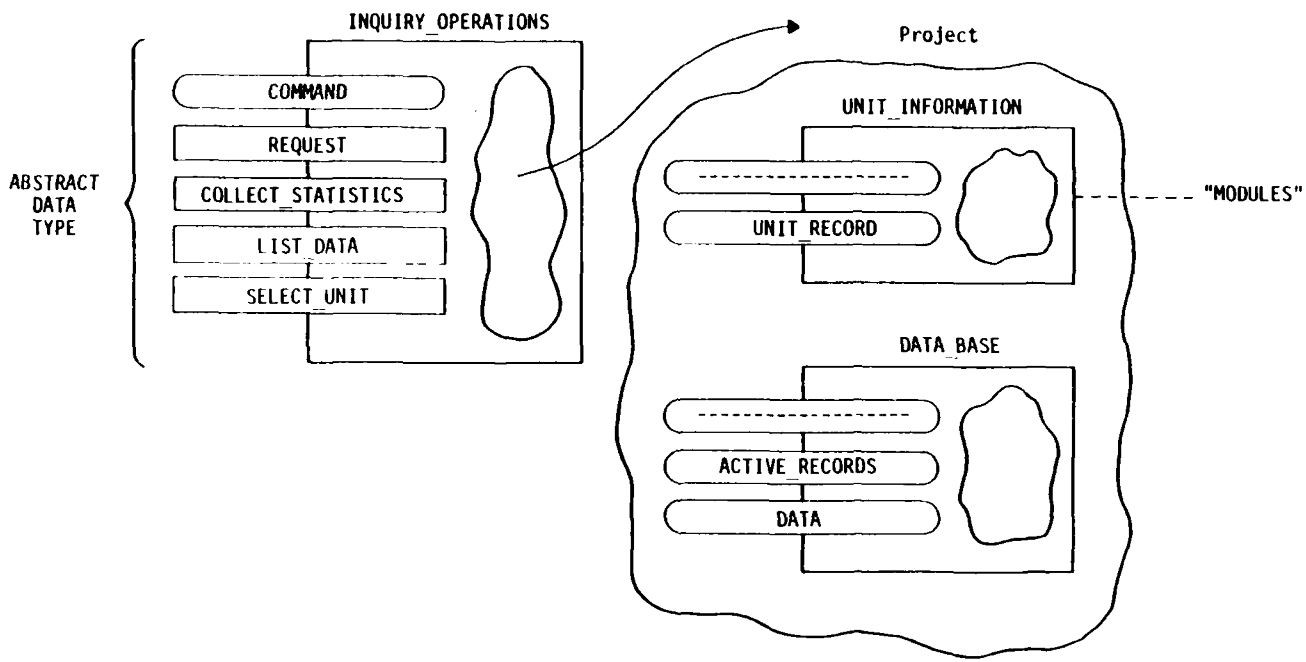
<u>COMMAND OPERATIONS</u>

- IMPLEMENTED WITH PROC. AND FTNS

OBJECT-ORIENTED DESIGN EXAMPLE

- ESTABLISH INTERFACES

```
with PROJECT;
use PROJECT;
package INQUIRY_OPERATIONS is
   type COMMAND is (COLLECT_STATISTICS, LIST_DATA, QUIT, SELECT_UNIT);
   function REQUEST return COMMAND;
   procedure COLLECT_STATISTICS;
   procedure LIST_DATA;
   procedure SELECT_UNIT;
end INQUIRY_OPERATIONS;
```



INSTRUCTOR NOTES

THIS METHOD HAS BEEN EXTENDED BY OTHERS (RAY BUHR FOR ONE) AND MAY BECOME AN IMPORTANT METHODOLOGY IN THE FUTURE DUE TO IT'S STRONG LINK TO Ada.

OBJECT-ORIENTED DESIGN SUMMARY

- RELIES ON DATA ABSTRACTIONS
- PRODUCES MODULES
- HIDES DESIGN DECISIONS
  - DATA STRUCTURE
  - ALGORITHM SELECTION
- GOAL $\longrightarrow$ GOOD MODULE SPECIFICATIONS
- BOTTOM UP APPROACH WORKS WELL ON SMALL PROBLEMS OR ON SEGMENTS OF A LARGER PROBLEM DECOMPOSED BY OTHER METHODS

INSTRUCTOR NOTES

THEME: THIS IS THE TRADITIONAL VIEW OF STRUCTURED DESIGN MANY OF THE OTHER METHODS HAVE EVOLVED FROM THIS METHODOLOGY.

PURPOSE: REVIEW THE BASIC CONCEPTS AND TECHNIQUES ASSOCIATED WITH STRUCTURED DESIGN.

REFERENCES:

MYERS, G. "RELIABLE SOFTWARE THROUGH COMPOSITE DESIGN"
PETROCELLI/CHARTER, NYC; 1975

CONSTANTINE, L., YOURDON, E. "STRUCTURED DESIGN"
PRENTICE-HALL, NJ; 1978

Section 16

STRUCTURED DESIGN METHODOLOGY

INSTRUCTOR NOTES

THIS METHODOLOGY IS ATTRIBUTED TO YOURDON AND CONSTANTINE. IT IS CONCERNED WITH MODULARIZATION, INTERCONNECTIVITY, AND THE FLOW OF DATA. WE ARE NOT INTERESTED IN THE DETAILS OF THE MODULE INTERNALS YET.

TRANSFORMS AND TRANSACTIONS WILL BE EXPLAINED SOON.

THE CONNECTIONS BETWEEN *MODULES SHOULD BE SIMPLE AND WELL DEFINED.*

OVERVIEW

- STRUCTURED DESIGN IS A METHOD FOR DEVELOPING A HIGH-LEVEL BLUEPRINT OF A SOFTWARE SYSTEM

- KEY CONCEPTS:
  - A SYSTEM CAN BE DESIGNED AFTER UNDERSTANDING ITS DATA FLOW
  - DESIGNS ARE BUILT AROUND EITHER THE MAJOR DATA TRANSFORMS OR TRANSACTION CENTERS OF A SYSTEM
  - DESIGN QUALITY CAN BE IMPROVED BY EVALUATING INTRAMODULE UNITY AND INTERMODULE CONNECTIONS

- USES A GRAPHICAL NOTATION SUPPLEMENTED WITH ANNOTATION TO EXPRESS
  - DATA FLOW (USES BUBBLE CHARTS)
  - STRUCTURAL DESIGN (USES STRUCTURE CHARTS)

INSTRUCTOR NOTES

THIS IS THE STRUCTURE OF THE REST OF THE LECTURE ON STRUCTURED DESIGN.

STRUCTURED DESIGN TOPICS

• DATA FLOW GRAPHS • STRUCTURE CHART SYNTAX • ANNOTATIONS • TRANSLATION PROCESS	OVERVIEW OF THE METHOD
• TRANSFORM - CENTERED DESIGN • TRANSACTION - CENTERED DESIGN	SPECIFIC TECHNIQUES

INSTRUCTOR NOTES

BUBBLE CHARTS AND DATA FLOW DIAGRAMS ARE OFTEN USED INTERCHANGEABLY.

DATA FLOW GRAPHS

(BUBBLE CHARTS)

- DATA FLOW GRAPHS ARE NETWORKS OF TRANSFORMATIONS CONNECTED BY DATA PIPELINES.
- TRANSFORMATIONS ARE FUNCTIONS THAT CHANGE THEIR INPUT DATA INTO OUTPUT.
- PIPELINES ARE CONTINUOUS STREAMS OF DATA, BE THEY INPUT OR OUTPUT.

BUBBLE CHART SYNTAX

PIPELINE	$\longrightarrow$	(DATA)
TRANSFORM	◯	(FUNCTION)

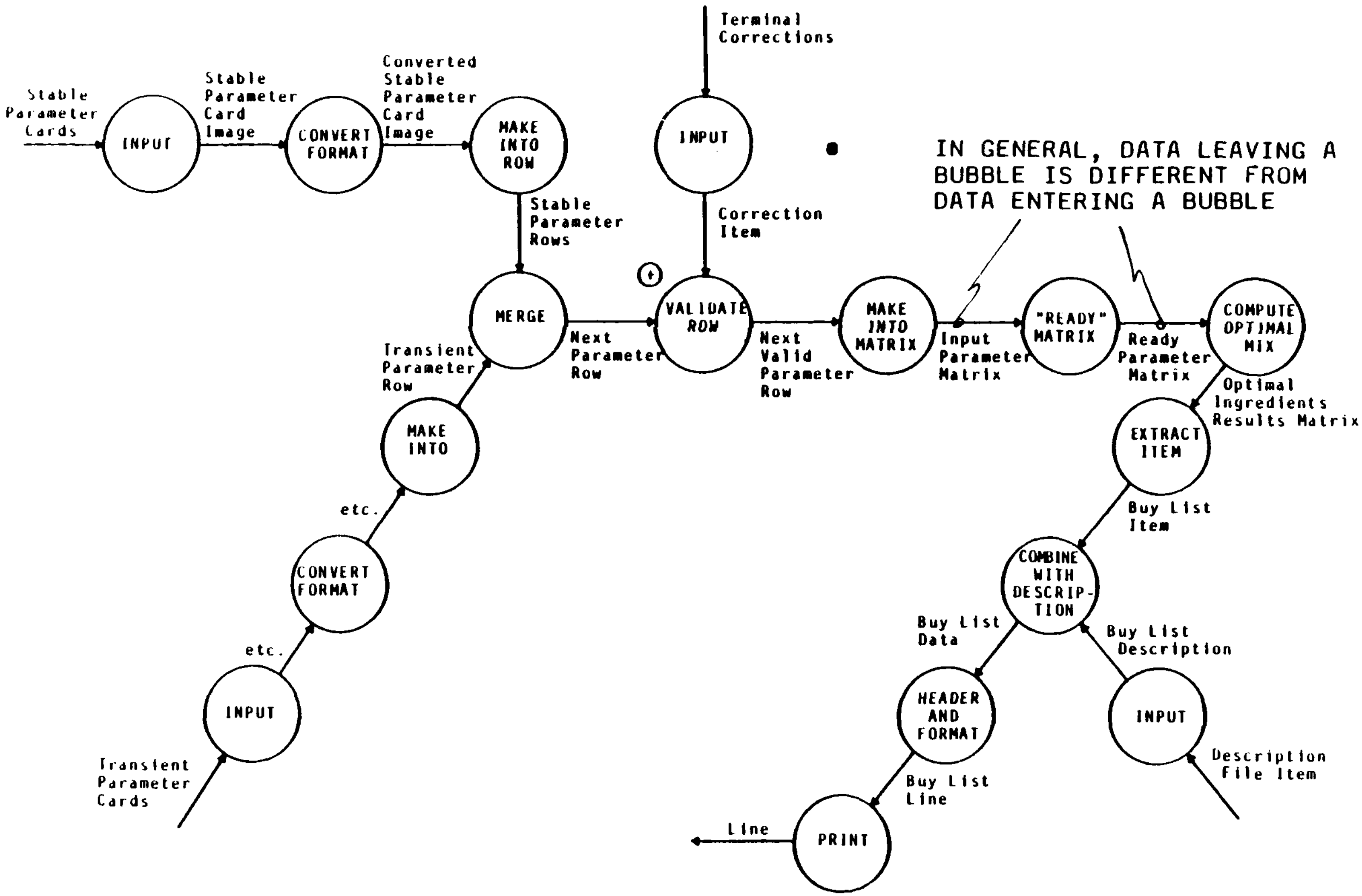
INSTRUCTOR NOTES

EXAMPLE REFERENCE: YOURDAN, E. AND CONSTANTINE, L.; "STRUCTURED DESIGN," PRENTICE-HALL, 1979.

NOTE THE EDP NATURE OF THE PROBLEM, BUT INDICATE TO THE CLASS THAT MAJOR PORTIONS OF MILITARY SYSTEMS HAVE SIMILAR DATA FLOWS (ONLY THE ARROWS AND BUBBLE LABELS WILL DIFFER).

DATA FLOW GRAPH

- ARROWS CONNECT BUBBLES TOGETHER INTO A NETWORK:



- IN GENERAL, DATA LEAVING A BUBBLE IS DIFFERENT FROM DATA ENTERING A BUBBLE

INSTRUCTOR NOTES

AT THIS LEVEL, EXCEPTIONAL CONDITIONS ARE NOT ADDRESSED.

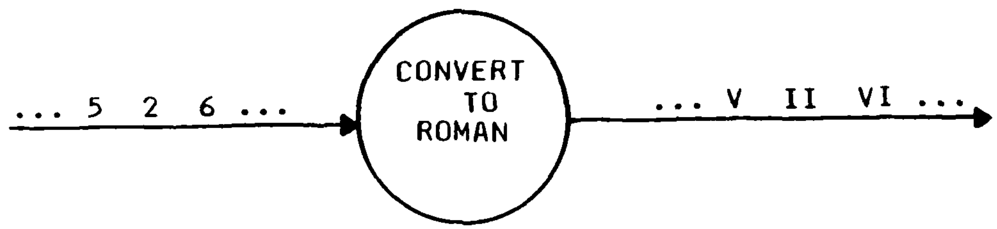
IN SOME REAL TIME SYSTEMS WHERE EXCEPTIONAL CONDITIONS DRIVE THE DESIGN THESE METHODS ARE LIMITED IN USEFULNESS.

DATA FLOW GRAPHS

- WHEN CONSTRUCTING DATA FLOW GRAPHS ...
  - START WITH FUNCTIONAL SOFTWARE REQUIREMENTS
  - ASSUME THE SYSTEM RUNS CONTINUOUSLY
  - WORK TOP-DOWN

- <u>CONTINUOUS SYSTEMS HAVE NO INITIALIZATION OR TERMINATION ISSUES</u>:

  (VERY RESTRICTIVE IN REAL TIME SOFTWARE)



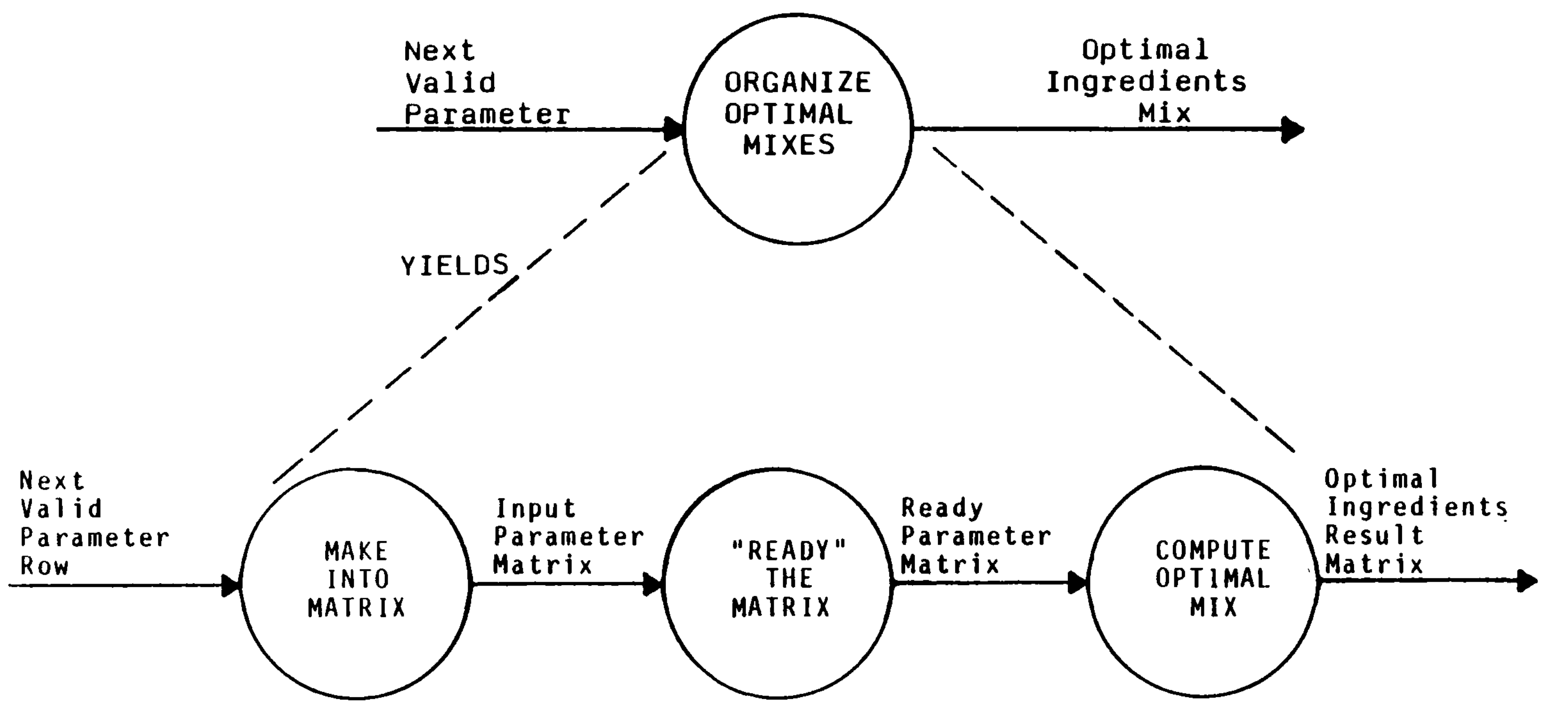
DATA PIPELINES ARE LIKE TICKER-TAPES, HAVING NO BEGINNING OR END.

INSTRUCTOR NOTES

RELATE DECOMPOSITION TO OTHER TECHNIQUES.

DATA FLOW GRAPHS

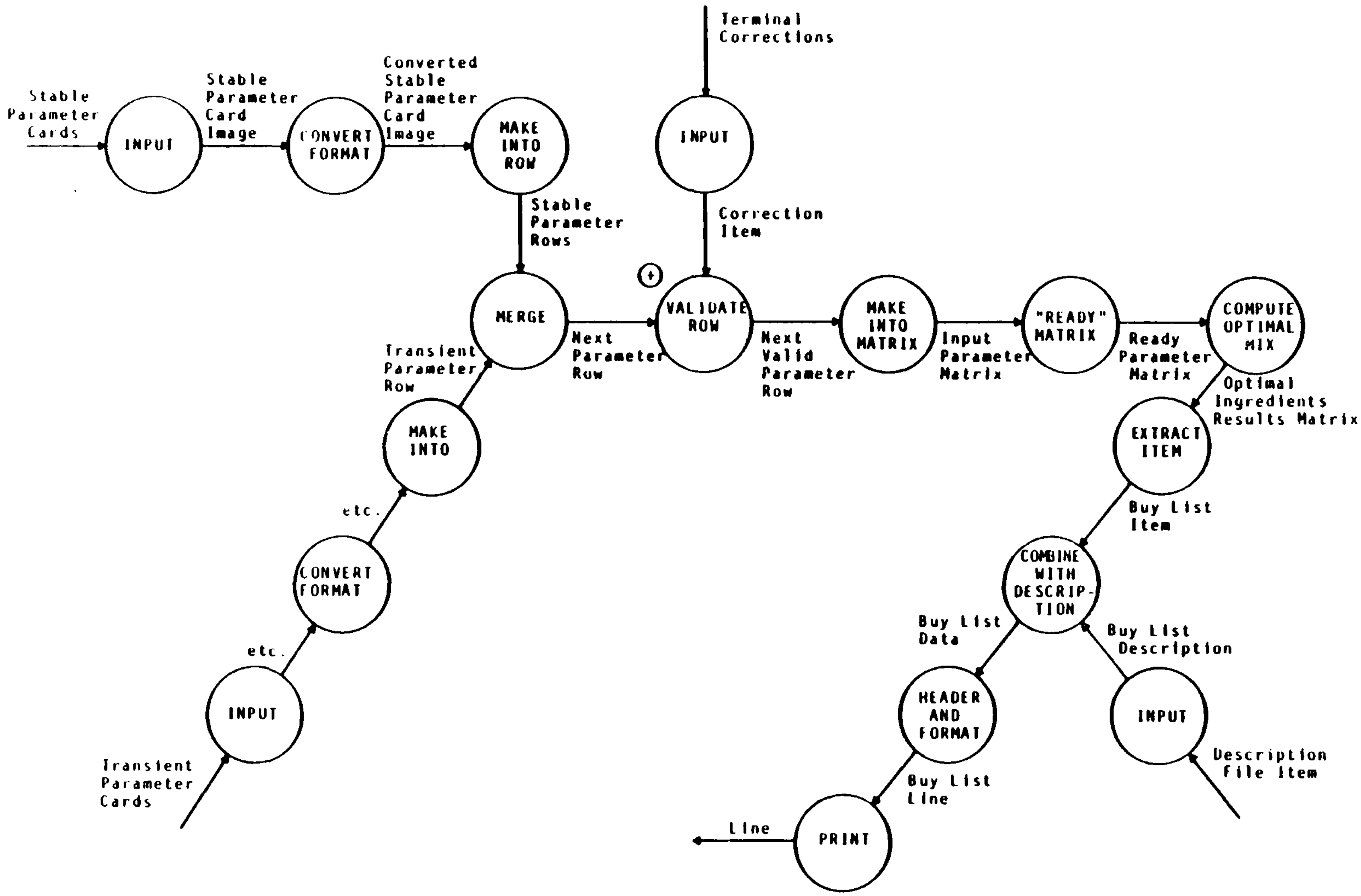
- TOP-DOWN DECOMPOSITION OF SOFTWARE REQUIREMENT



INSTRUCTOR NOTES

DATA FLOW GRAPH

- EACH BUBBLE ASSUMES OTHER BUBBLES WILL GENERATE CONTINUOUS DATA STREAMS ...



INSTRUCTOR NOTES

DATA FLOW GRAPHS SUMMARY

- DATA FLOW GRAPHS CAN BE DERIVED FROM SOFTWARE REQUIREMENTS BY A DECOMPOSITION PROCESS

- DATA FLOW GRAPHS ARE NETWORKS OF TRANSFORMS CONNECTED BY PIPELINES OF CONTINUOUS DATA STREAMS

- THEY DO <u>NOT</u> SHOW ...
  - DATA DECOMPOSITION
  - DATA HIERARCHY
  - PROCEDURAL DETAILS
  - CONTROL LOGIC
  - ACTIVATION OR TERMINATION CONDITIONS

INSTRUCTOR NOTES

THESE SYMBOLS ARE NOT LANGUAGE SPECIFIC. POINT OUT DIFFERENCES BETWEEN THE "FILLED" ARROW AND "EMPTY" ARROW.

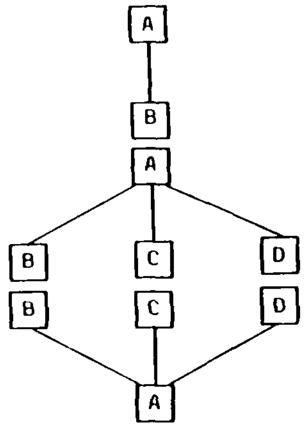
STRUCTURE CHART

- THE BASIC SYNTAX, USED MOST OFTEN COMPRISES FOUR SYMBOLS ...

MODULE	(SOFTWARE FUNCTION)
INTERFACE	(THE PLACE WHERE DATA IS PASSED BETWEEN MODULES)
DATA	(APPLICATION)
	(CONTROL)

- BASIC STRUCTURAL ASPECTS

MODULE COMBINATIONS



MODULE A CALLS MODULE B.

MODULE A CALLS MODULES B, C, AND D.

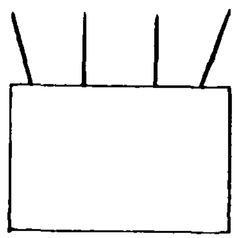
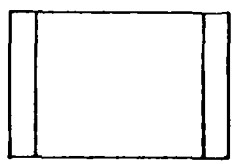
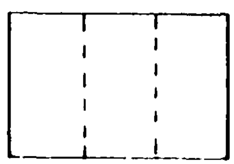
MODULE A IS CALLED BY B, C, AND D.

INSTRUCTOR NOTES

MAKE SURE CLASS UNDERSTANDS THESE SYMBOLS.

STRUCTURE CHART

- HERE'S MORE DETAIL ABOUT BASIC SYNTACTIC UNITS ...

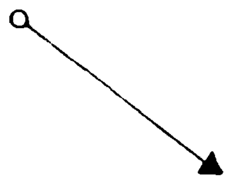
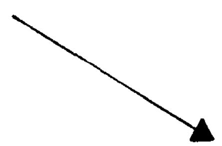
SYMBOL	MEANING
	MODULE THAT IS CALLED FROM SEVERAL PLACES IN STRUCTURE.
	PRE-EXISTING MODULE OR COMMON FUNCTION (ALTERNATIVE SYMBOL FOR 1).
	MODULE WITH MULTIPLE ENTRY POINTS (THIS ONE HAS THREE).

INSTRUCTOR NOTES

THE THIRD ARROW TYPE SHOULD BE ELIMINATED AS THE DESIGN IS REFINED.

STRUCTURE CHART

- STILL MORE ...

SYMBOL	MEANING
	AN ARROW WITH A SOLID TAIL DENOTES AN INTERFACE THAT IS AN ELEMENT OF CONTROL (FLAG, SWITCH, INDICATOR).
	AN ARROW WITH A CIRCLE ON IT'S TAIL DENOTES AN INTERFACE THAT IS AN ELEMENT (E.G. STRUCTURE, ARRAY) OF APPLICATION DATA.
	AN ARROW WITH A PLAIN TAIL COULD BE CONTROL, DATA, OR BOTH.

IT IS RECOMMENDED THAT A DESIGNER DISTINGUISH BETWEEN CONTROL AND DATA.

INSTRUCTOR NOTES

IT IS IMPORTANT TO UNDERSTAND THAT MODULES MAY CONTAIN OTHER MODULES. THE WHOLE SYSTEM IS A "MODULE."

STRUCTURE CHART SYNTAX

- THE SYNTAX IS GEARED AROUND <u>MODULES</u> AND THEIR <u>INTERFACES</u>

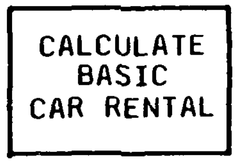
- FROM THE VIEWPOINT OF STRUCTURED DESIGN, A MODULE IS ...
  - A NAMED GROUP OF PROGRAM STATEMENTS WITH FOUR BASIC ATTRIBUTES:

1	INPUT OUTPUT	(INFORMATION IT USES) AND (INFORMATION IT PRODUCES)	THIS IS THE <u>OUTSIDE</u> VIEW OF THE MODULE
2	FUNCTION	(WHAT IT DOES TO ITS INPUT TO PRODUCE ITS OUTPUT)	
3	LOGIC	(THE EXECUTABLE CODE IT NEEDS TO CARRY OUT ITS FUNCTION)	THIS IS THE <u>INSIDE</u> VIEW OF THE MODULE
4	DATA	(PRIVATE DATA THAT IT ALONE REFERS TO)	

INSTRUCTOR NOTES

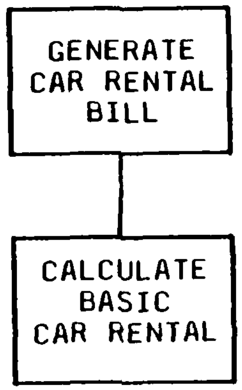
STRUCTURE CHART

- <u>GRAPHIC REPRESENTATION OF A MODULE</u>



  - NOTICE THAT THE NAME OF THE MODULE IS A PRECISE STATEMENT OF ITS FUNCTION.

- <u>REPRESENTATION OF A MODULE CALL</u>



Boss (Caller)

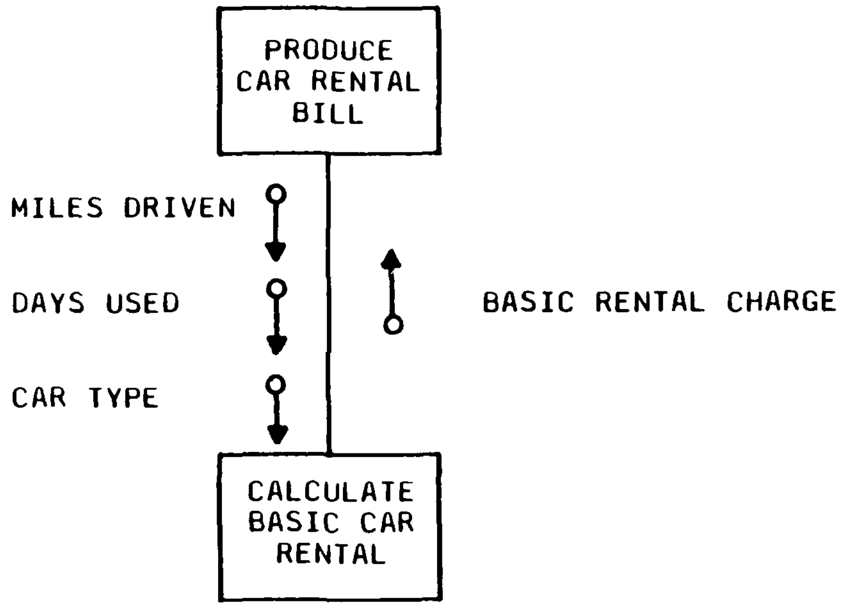
Subordinate (Called)

  - THE | (INTERFACE LINE) MEANS BOTH THE CALL <u>AND</u> THE RETURN.

INSTRUCTOR NOTES

THE DIRECTION OF DATA FLOW IS THE DIRECTION OF THE ARROW.

- <u>MODULE COMMUNICATION</u>:



  - THE MODULE <u>PRODUCE CAR RENTAL BILL</u> SENDS THREE PIECES OF DATA TO THE MODULE <u>CALCULATE BASIC CAR RENTAL</u>:

    MILES DRIVEN, DAYS USED AND CAR TYPE

  - THE MODULE <u>CALCULATE BASIC CAR RENTAL</u> SENDS BACK ONE PIECE OF DATA TO THE MODULE <u>PRODUCE CAR RENTAL BILL</u>:

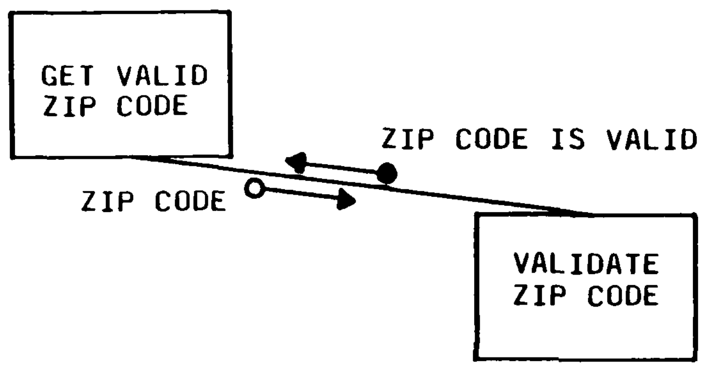
    BASIC RENTAL CHARGE

INSTRUCTOR NOTES

CONTROL DATA IN GENERAL MODIFIES THE WAY IN WHICH DATA IS TRANSFORMED WITHIN THE MODULE.

STRUCTURE CHART

- AN EXAMPLE OF PASSING CONTROL DATA.



- <u>ZIP CODE IS VALID</u> IS A PIECE OF CONTROL INFORMATION.
- ○——▶ REPRESENTS DATA

  ●——▶ REPRESENTS A FLAG (SWITCH)
- THE DIRECTION OF THE ARROW SHOWS THE DIRECTION OF THE INFORMATION FLOW.

INSTRUCTOR NOTES

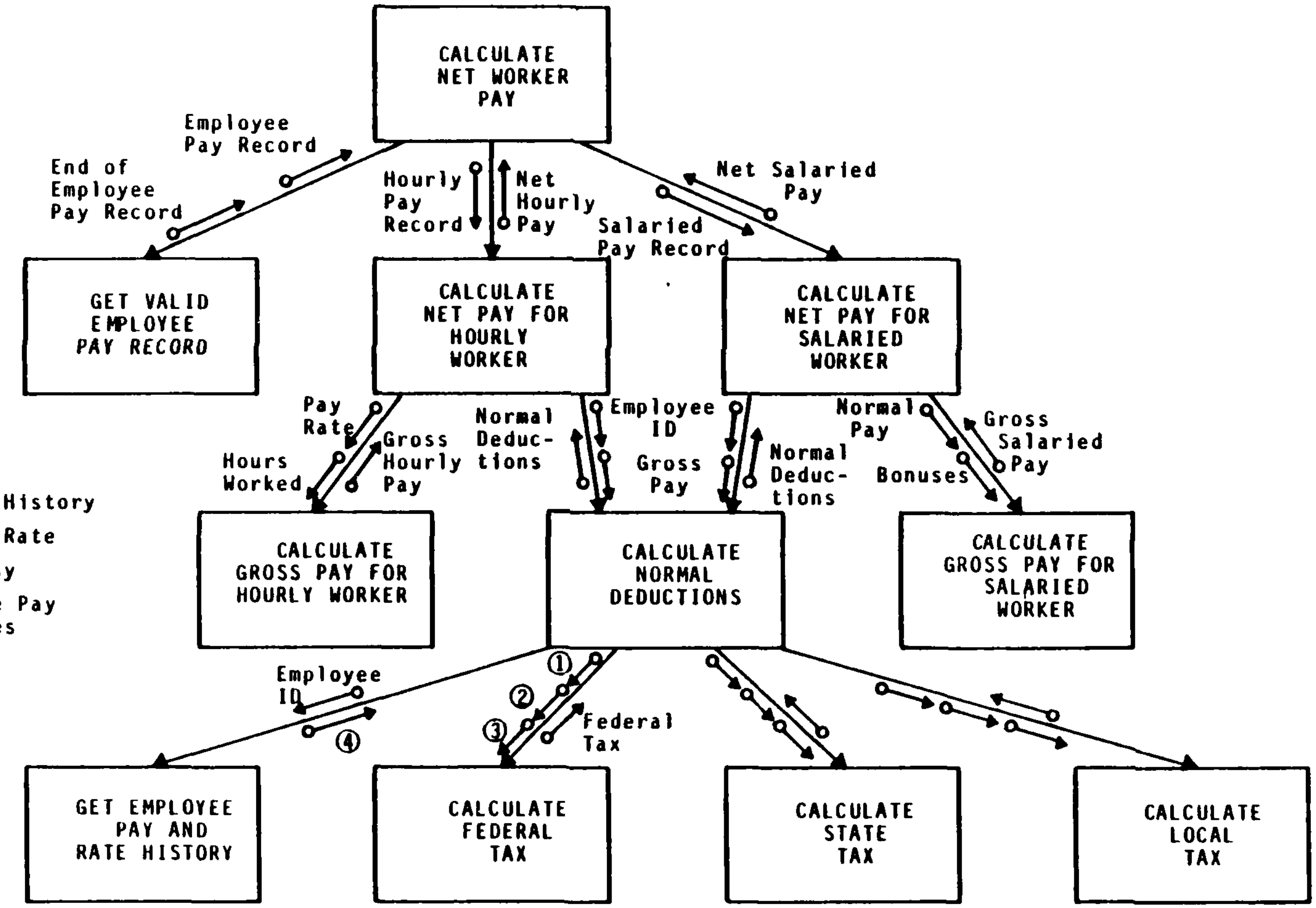
WHETHER THE CALLED MODULES ARE PHYSICALLY CONTAINED IN THE CALLER MODULE OR ANOTHER MODULE IS IMMATERIAL AT THIS POINT.

WITHOUT THE ARROWS, THIS IS CALLED A CALLING HIERARCHY DIAGRAM. AT THE MINIMUM THERE SHOULD BE ONE FOR EVERY SYSTEM.

THIS IS A GOOD TECHNIQUE FOR ***DESCRIBING*** *THE DESIGN:* IT WILL SUPPORT ANALYSIS OR DESIGN QUALITY.

STRUCTURE CHART

- THE TOTAL EFFECT IS A HIERARCHY OF MODULES ...



① Federal History
② Federal Rate
③ Gross Pay
④ Employee Pay and Sales History

INSTRUCTOR NOTES

CONVENTION SAYS CALLED IN ORDER FROM LEFT TO RIGHT. IT IS NOT NECESSARILY A RULE.

SUMMARY OF THE STRUCTURE CHART

- STRUCTURE CHARTS SHOW:
  - PARTITIONING
  - HIERARCHY AND ORGANIZATION
  - COMMUNICATION
  - FUNCTIONS

- STRUCTURE CHARTS DON'T SHOW:
  - CALLING <u>SEQUENCE</u>
  - PROCEDURE
  - INTERNAL DATA
  - ACTIVATION AND TERMINATION CONDITIONS OR EVENTS

INSTRUCTOR NOTES

THE BASIC SYNTAX IS NOT SUFFICIENT TO EXPRESS THE STRUCTURE OF ALL PROBLEMS.

ANNOTATIONS

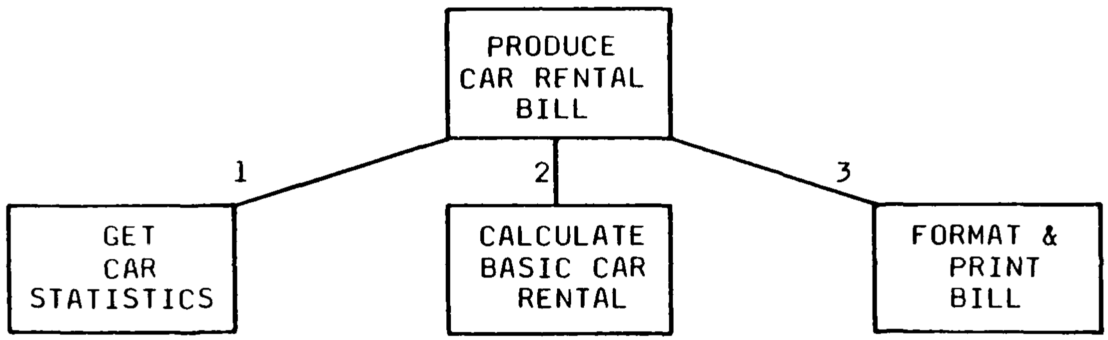
- THE BASIC SYNTAX HAS EXTENSIONS, WHICH ANNOTATE THE STRUCTURE CHART TO:
  - ORGANIZE INTERFACE DESCRIPTIONS
  - DENOTE CONDITIONAL CALLING CONTROL TRANSFERS AND ITERATION

INSTRUCTOR NOTES

WHEN THERE ISN'T ENOUGH ROOM FOR ALL THE DATA, SUBSTITUTE A CHART. ALTHOUGH NOT AS COMMUNICATIVE, IT IS NOTATIONALLY CONVENIENT.

ANNOTATIONS

- <u>TO ORGANIZE INTERFACE DESCRIFTIONS</u>: NUMBERS ARE USED TO MARK EACH LINE AND A TABLE DESCRIBES THE DATA TRANSFERS ...



	IN	OUT
1	~	MILES DRIVEN DAYS USED CAR TYPE DRIVER NAME
2	MILES DRIVEN DAYS USED CAR TYPE	BASIC RENT CHARGE
3	DRIVER NAME BASIC RENT CHARGE	~

INSTRUCTOR NOTES

THE EXAMPLE IS "GOTO AN ENTRY POINT AT THE BEGINNING OF MODULE B". THIS IS NOT RECOMMENDED.

UNCONDITIONAL IS DEFAULT.

CONDITIONAL CALL SYMBOLIZES "SOMETIMES." THE FREQUENCY OF CALLS IS A FINER DETAIL; NOT *OFTEN USED DURING DESIGN*, IT MAY BECOME IMPORTANT IF "SOMETIMES" IS <u>VERY</u> INFREQUENT OR <u>VERY</u> FREQUENT.

AGAIN, REPETITION MAY BE IMPORTANT; ESPECIALLY IF MODULE A CALLS MODULE B ONE MILLION TIMES. READ AS "IN A GIVEN EXECUTION OF MODULE A ..."

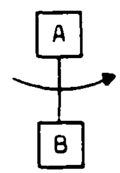
ANNOTATIONS

SYMBOL — MEANING

(1)

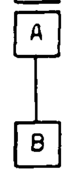
MODULE A CALLS MODULE B ONLY ONCE.

(2)



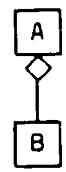
MODULE A CALLS MODULE B MANY TIMES.

(3)



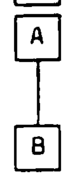
MODULE A CALLS MODULE B *UNCONDITIONALLY*.

(4)



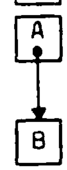
MODULE A CALLS MODULE B ONLY SOMETIMES.

(5)



MODULE A CALLS MODULE B WHO THEN RETURNS BACK TO MODULE A (STANDARD CALLING SEQUENCE).

(6)



MODULE A TRANSFERS CONTROL TO MODULE B UNCONDITIONALLY (WITHOUT RETURN).

INSTRUCTOR NOTES

◇ FROM A TO C - A INVOKES C SOMETIMES.

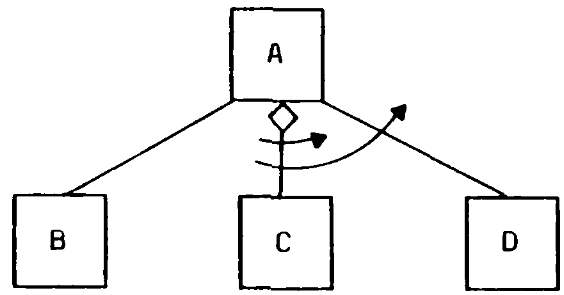
ON C - A DECIDES TO INVOKE C MANY TIMES.

ON C AND D - A TRIES TO INVOKE BOTH C AND D MANY TIMES.

EMPHASIZE THAT LOCATION OF BOXES ON STRUCTURE CHART DOES NOT IMPLY THE SEQUENCE OF INVOCATION.

ANNOTATIONS

- COMPOUND ANNOTATION ALLOWS COMPLEX CALLING SEQUENCE TO BE REPRESENTED.



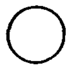
  - MODULE A CALLS B THEN CONDITIONAL CALLS C THEN CALLS D. MODULE C AND D CAN BE CALLED MULTIPLE TIMES

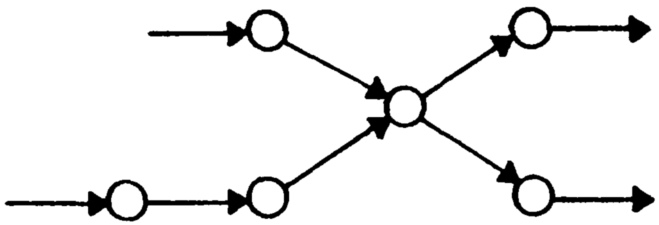
INSTRUCTOR NOTES

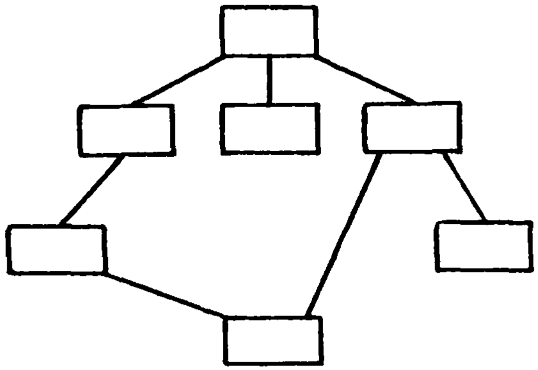
THERE IS NO ONE COOKBOOK TECHNIQUE TO GET A GOOD STRUCTURE CHART FROM A DATA FLOW GRAPH. HOWEVER, BUBBLES AND BOXES BOTH REPRESENT ACTIVITIES WHILE ARROWS AND INTERFACE LINES REPRESENT DATA.

TRANSLATION PROCESS

- DATA FLOW GRAPHS ARE DRAWN FIRST: ⟹ STRUCTURE CHARTS ARE THEN DRAWN, BASED ON THE DATA FLOW GRAPHS.

- IN GENERAL:  ⟹ 





- THE TRANSLATION PROCESS SOMETIMES DOESN'T CREATE ONE BOX PER BUBBLE BECAUSE:
  - COMMONLY REQUIRED SERVICES ARE CLUSTERED INTO ONE OR A VERY FEW BOX(ES)
  - A BUBBLE MAY BE TOO GENERAL, AND REQUIRES MORE BOXES TO DESCRIBE ITS FUNCTION

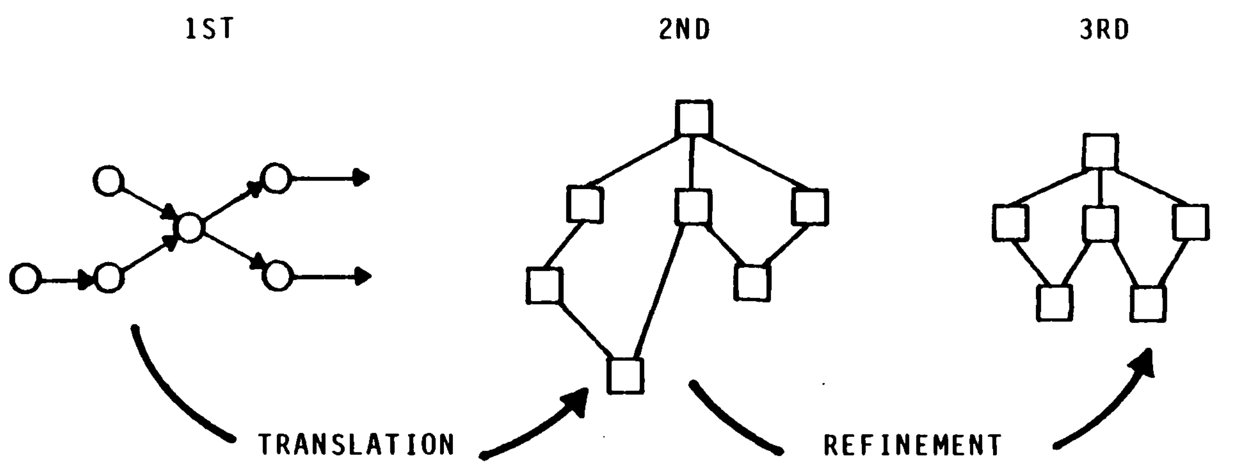
INSTRUCTOR NOTES

THESE 2 TECHNIQUES ARE TO <u>HELP</u> IN THE TRANSLATION. THE MAJOR TECHNIQUE USED IS TRANSFORM CENTERED DESIGN.

TRANSLATION PROCESS

- TWO TECHNIQUES GUIDE THE DESIGNER DURING THE GENERAL PROCESS:
  - TRANSFORM-CENTERED
  - TRANSACTION-CENTERED

- AFTER THE TRANSLATION PROCESS, THE STRUCTURE CHART IS EVALUATED AND REFINED ACCORDING TO METRICS ...

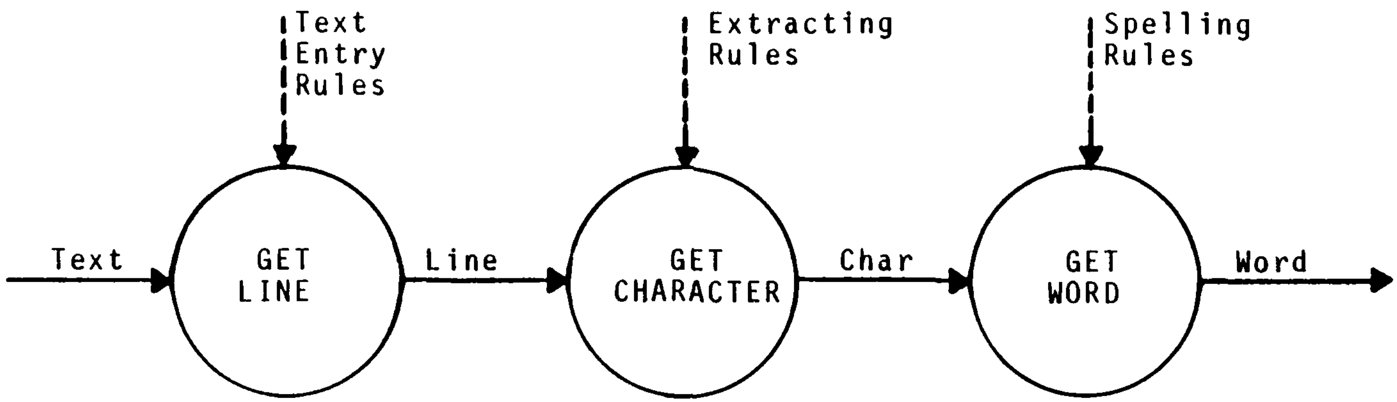


INSTRUCTOR NOTES

POINT OUT THAT THESE RULES GO FROM VERY SPECIFIC TO VERY GENERAL AS THE DATA GOES THROUGH SUCCESSIVE LEVELS OF TRANSFORMATION.

TRANSFORM-CENTERED DESIGN

- ALL BUBBLES IN A DATA FLOW DIAGRAM TRANSFORM DATA, BUT THE TRANSFORMATION RULES OF EACH BUBBLE ARE AT DIFFERENT LEVELS OF GENERALIZATION



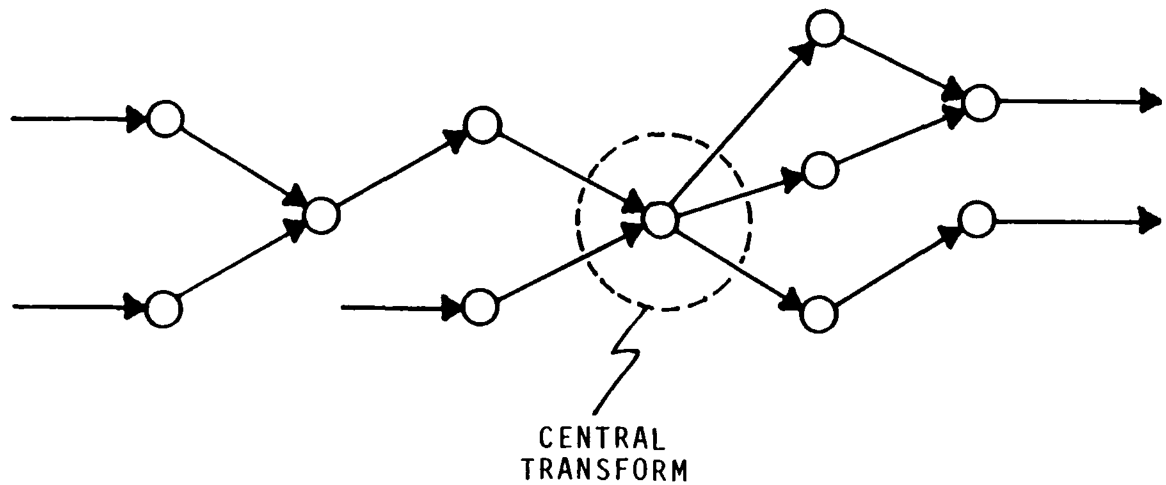
- HAS THREE DIFFERENT LEVELS OF RULES:
  - TEXT ENTRY - PHYSICAL I/O CONCERNS
  - CHARACTER EXTRACTION - CONCERNS ABOUT STORING LINES OF CHARACTERS
  - SPELLING - HOW WORDS ARE FORMED

INSTRUCTOR NOTES

FIRST, FIND THE MAJOR DATA TRANSFORMS.

TRANSFORM-CENTERED DESIGN

- TRANSFORM-CENTERED DESIGN STRATEGY SAYS:
  - "DESIGNS CAN BE BUILT AROUND THE MAJOR DATA TRANSFORMATION RULES OF THE APPLICATION."
  - THE HIGHEST LEVEL HAS THE MOST GENERAL RULE:

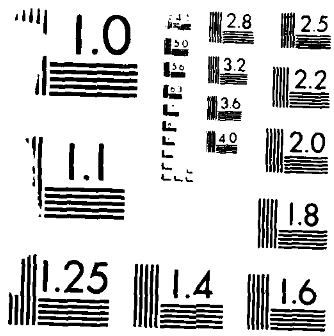


INSTRUCTOR NOTES

EMPHASIZE THE ALMOST COOKBOOK NATURE OF THE PROCEDURES.

AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2 (U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506 4/6

UNCLASSIFIED F/G 5/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

TRANSFORM-CENTERED DESIGN

TRANSFORM-CENTERED DESIGN PROCEDURE:

1. DRAW THE DATA FLOW GRAPH.
2. LABEL WHERE THE HIGHEST TRANSFORMS ARE.
3. MAKE THE HIGHEST TRANSFORMS THE TOP OF THE DESIGN. (I.E. TOP OF THE STRUCTURE CHART)
4. EVERYTHING TO THE LEFT OF THE TRANSFORMS IS CONSIDERED INPUT (AFFERENT).
5. EVERYTHING TO THE RIGHT OF THE TRANSFORMS IS CONSIDERED OUTPUT (EFFERENT).

INSTRUCTOR NOTES

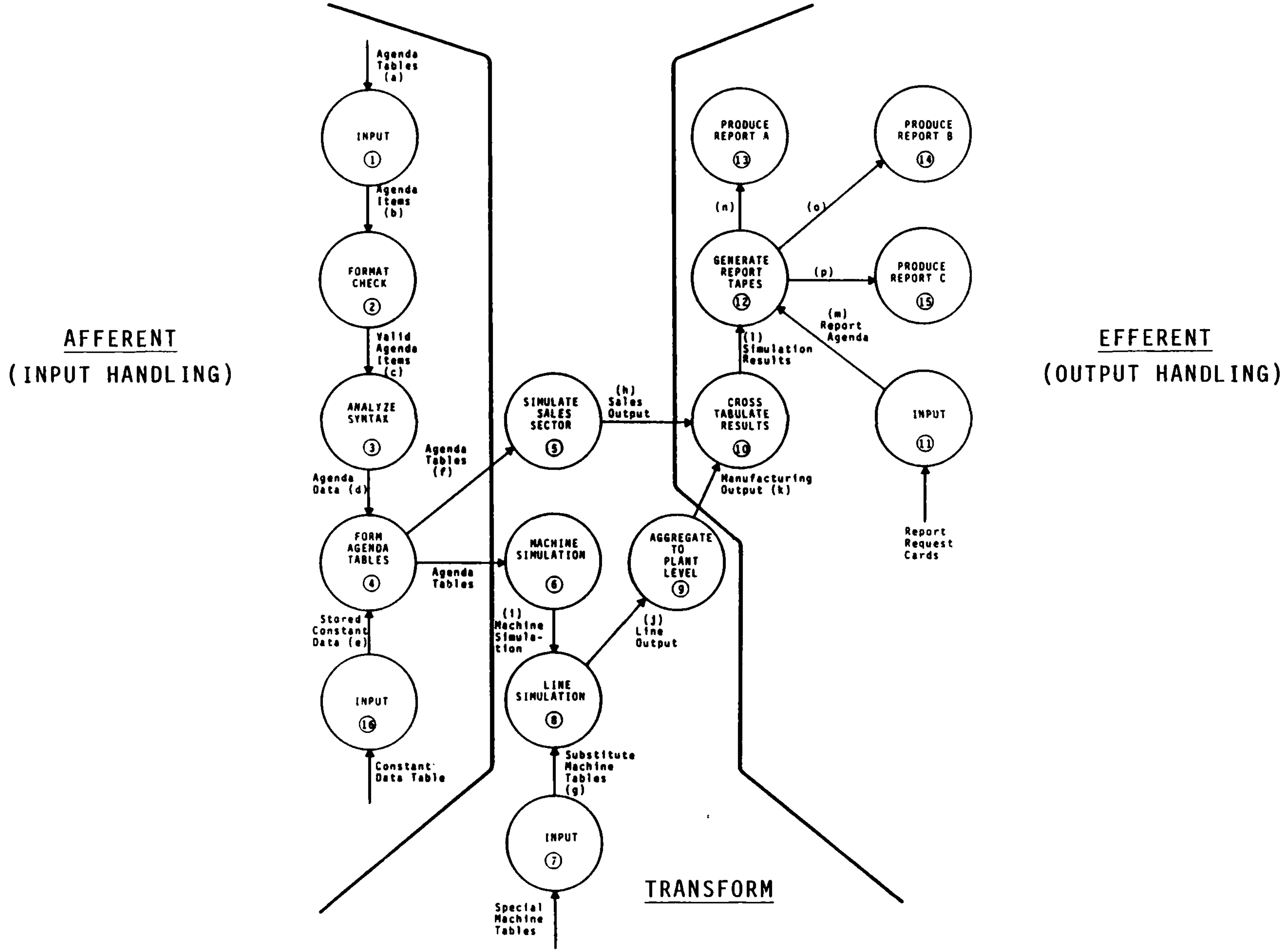
BUBBLES 7 AND 11 ARE INPUTS OF <u>RULES</u> WHICH ARE <u>NOT</u> APPLICATION DATA. SO THEY REMAIN IN THEIR RESPECTIVE CATEGORIES (TRANSFORM AND EFFERENT RESPECTIVELY).

WALK THROUGH DATA FLOW, POINTING OUT WHEN FUNCTIONS:

(1) STOP BEING CONCERNED WITH INPUT,

(2) START BEING CONCERNED WITH OUTPUT

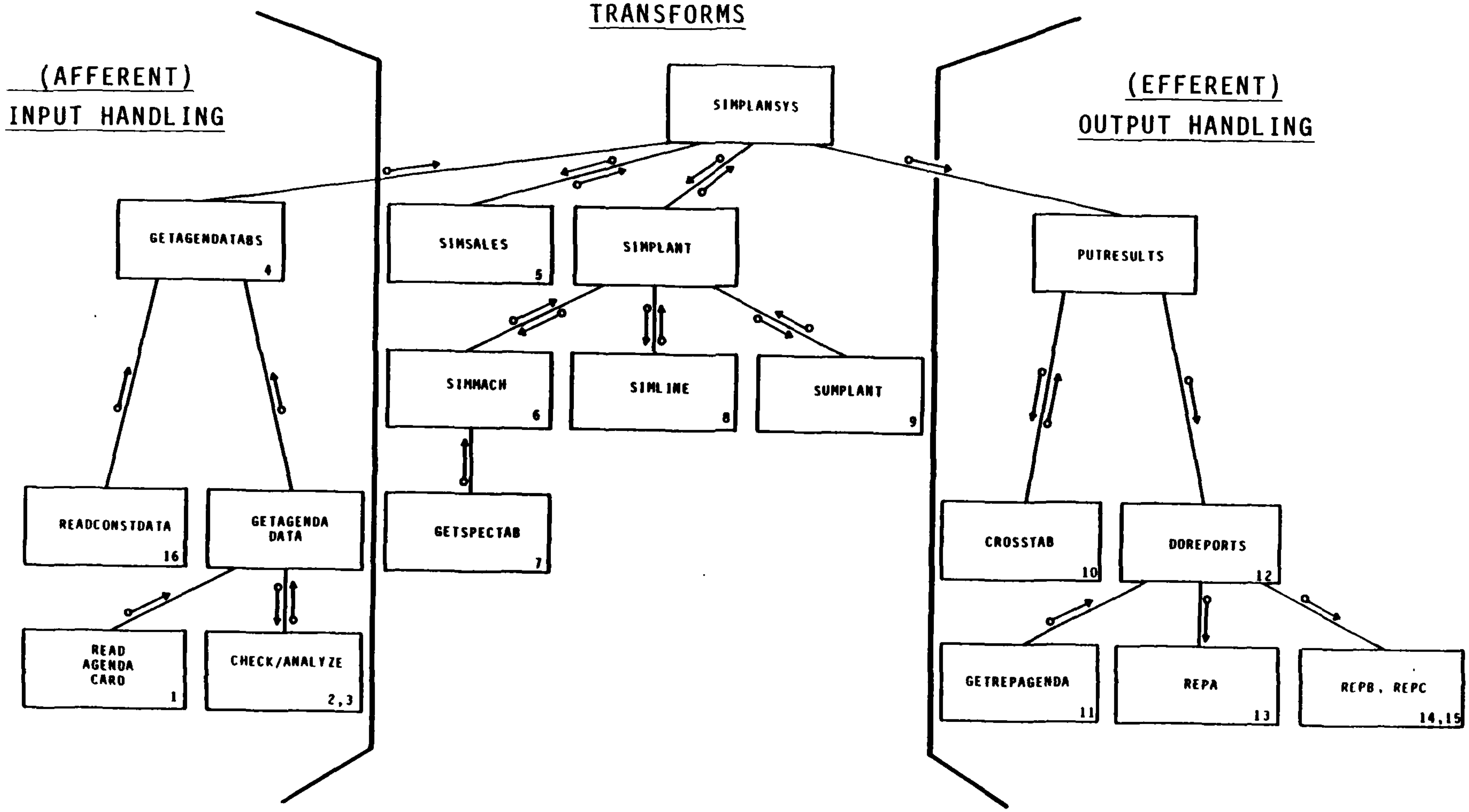
TRANSFORM-CENTERED DESIGN



INSTRUCTOR NOTES

SHOW HOW THE DESIGN'S STRUCTURE MATCHES THE AFFERENT-EFFERENT DIVISION MADE ON THE DATA FLOW DIAGRAM.

TRANSFORM-CENTERED DESIGN



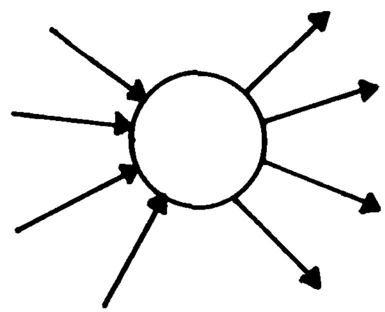
INSTRUCTOR NOTES

TRANSACTION-CENTERED DESIGN IS THE PREFERRED METHOD.

TRANSACTION-CENTERED DESIGN

- SOMETIMES, SOME BUBBLES IN A DATA FLOW GRAPH ARE CENTERS FOR LARGE VOLUMES OF TRANSACTIONS.

- THE PATTERN TO LOOK FOR IS:



- TRANSACTION-CENTERED DESIGN STRATEGY SAYS:
  - "DESIGNS CAN BE BUILT AROUND A "RECEIVE AND DISPATCH" CENTER."

INSTRUCTOR NOTES

THE STRUCTURE FOR A TRANSACTION CENTERED DESIGN IS:

1. GET
2. ANALYZE
3. DISPATCH
4. PROCESS

(CONTRAST TO TRANSFORM CENTERED DESIGN WHICH IS:

1. INPUT
2. TRANSFORM
3. OUTPUT)

TRANSACTION-CENTERED DESIGN

- <u>TRANSACTION-CENTERED DESIGN PROCEDURE</u>:

    1. DRAW THE DATA FLOW GRAPH.

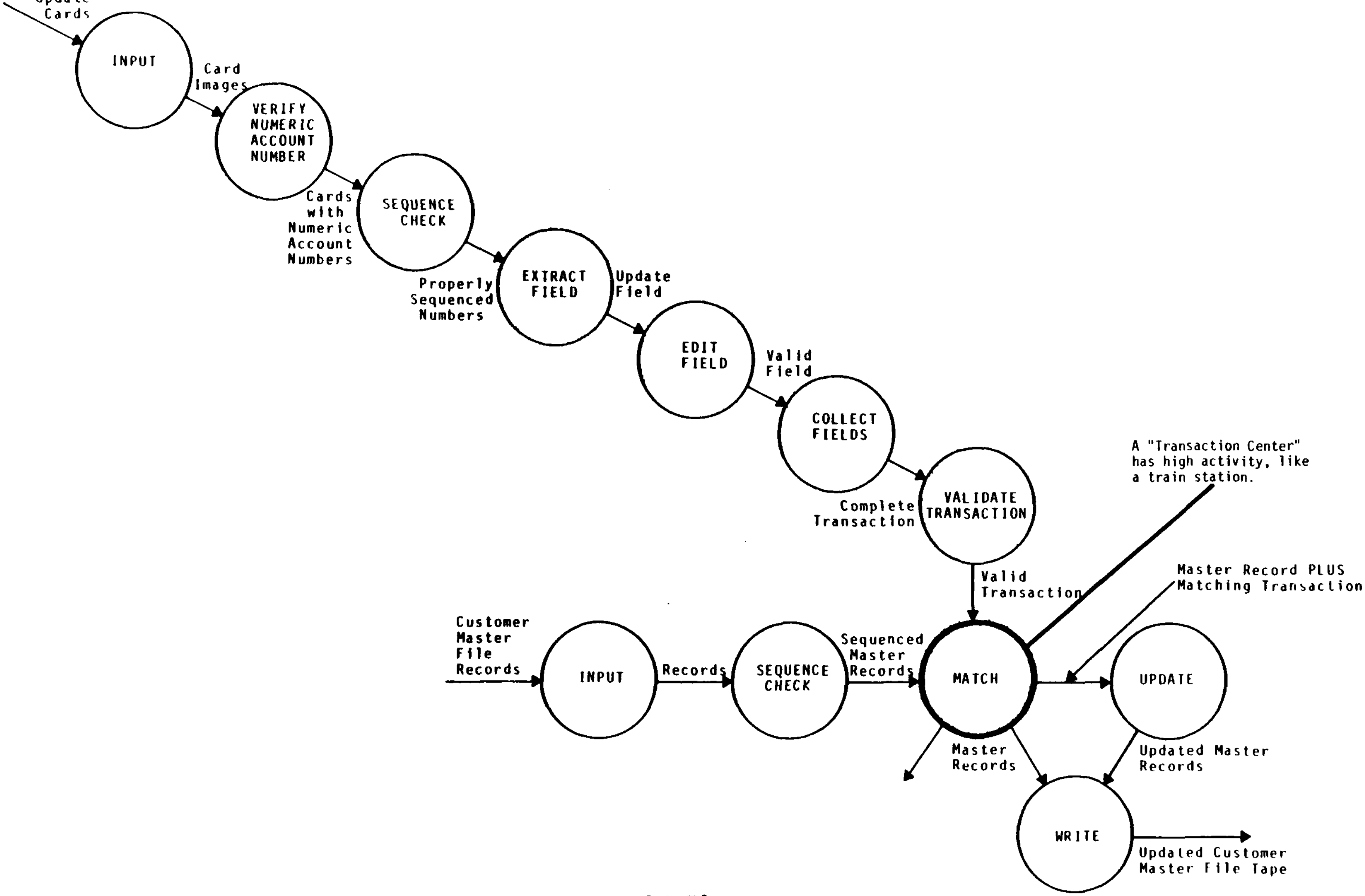
    2. FIND THE BUBBLE WITH THE HIGHEST VOLUME OF TRANSACTION DISPATCHING <u>THAT IS ALSO</u> A HIGH-LEVEL TRANSFORM.

    3. MAKE THIS BUBBLE THE TOP OF THE DESIGN. (I.E. TOP OF STRUCTURE CHART)

INSTRUCTOR NOTES

POINT OUT THAT OFTEN, TRANSACTION CENTERS JUST ACCEPT TRANSACTIONS AND LET THEM PASS ACCORDING TO SOME KIND OF COMPARISON LOGIC.

TRANSACTION-CENTERED DESIGN

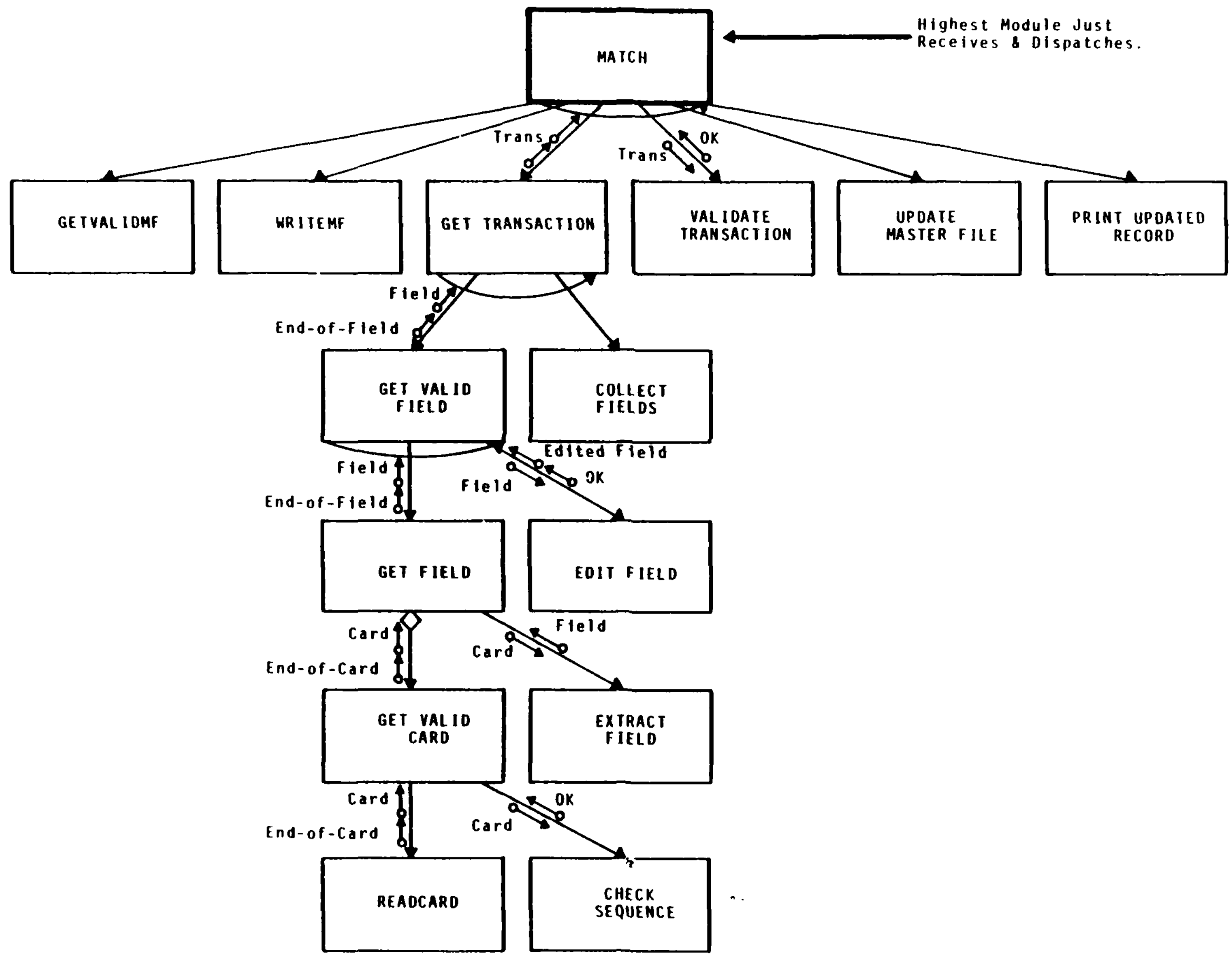


INSTRUCTOR NOTES

POINT OUT THE SHAPE OF THE RESULTING DESIGN:

PANCAKE TOP AND TOWER BOTTOM. STATE THAT WE'LL COVER METRICS IN NEXT CHAPTER THAT HELP US EVALUATE WHETHER THIS DESIGN IS GOOD OR NOT.

TRANSACTION-CENTERED DESIGN



INSTRUCTOR NOTES

ASK WHY REAL TIME APPLICATIONS DON'T WORK WELL.

STRUCTURED DESIGN SUMMARY

- PROVIDES METHODS FOR REPRESENTING SOFTWARE ARCHITECTURAL DESIGN
- PROVIDES GUIDANCE AS TO MANNER IN WHICH MODULARITY IS DETERMINED
- WORKS VERY WELL ON MEDIUM SIZED APPLICATIONS WHICH RESULT IN SEQUENCE PROGRAMS
- WORKS MARGINALLY IN REAL TIME APPLICATIONS

INSTRUCTOR NOTES

THEME: DESIGNS SHOULD MIRROR DATA STRUCTURE OF THE APPLICATION.

PURPOSE: TO SUMMARIZE A VERY POPULAR (AT LEAST IN EUROPE) DESIGN METHOD.

REFERENCE: JACKSON, M. "PRINCIPLES OF PROGRAM DESIGN" ACADEMIC PRESS, LONDON: 1975

JACKSON, M. "SYSTEM DEVELOPMENT" PRENTICE-HALL, NJ; 1983

Section 17

JACKSON METHODOLOGY

INSTRUCTOR NOTES

THIS TECHNIQUE WAS INTENDED FOR BUSINESS APPLICATIONS. THE JSP CAME FIRST (EARLY 70's), WITH JSD COMING ABOUT IN THE EARLY 80's. IN ENGLAND THERE EXIST TECHNICAL GROUPS WHOSE SPECIFIC INTEREST IS THIS METHODOLOGY.

WE WILL CONCENTRATE ON JSP AND ITS PRINCIPLES; AND ONLY A LITTLE OF JSD. THE PLANNED EXERCISE SHOWS JSP IN ACTION.

OVERVIEW

- CONSISTS OF TWO MAJOR ASPECTS

JACKSON DESIGN METHODOLOGY

JACKSON STRUCTURED PROGRAMMING (JSP)	JACKSON SYSTEM DEVELOPMENT (JSD)

- JACKSON SYSTEM DEVELOPMENT EVOLVED FROM JACKSON STRUCTURED PROGRAMMING

INSTRUCTOR NOTES

JACKSON'S VIEW OF REALITY IS A SET OF PROCESSES TRANSFORMING DATA STRUCTURES WHICH ARE INHERENT IN THE PROBLEM. THIS DATA IS USED TO MODEL THE PROBLEM.

NOTICE THE SUPPORT LENT TO BUSINESS SYSTEMS WHICH ARE DATA BASE ORIENTED.

JACKSON SYSTEM DEVELOPMENT OVERVIEW

- WAS DEVELOPED IN THE EARLY 80's
- IS SUITABLE FOR ARCHITECTURAL DESIGN OF LARGE SYSTEMS
- JACKSON SYSTEM DEVELOPMENT CHARACTERISTICS
  - IS NOT TOP-DOWN
  - MODELS REALITY, NOT FUNCTION
  - FOCUSES ON DATA STRUCTURES AND RELATED TIMING CONSTRAINTS.
  - IDENTIFIES DATA STREAMS AND STATE INFORMATION EARLY IN DESIGN.
  - BREAKS A SYSTEM INTO <u>PROCESSES</u> (THAT COULD BE RUN IN PARALLEL).
- JACKSON SYSTEM DEVELOPMENT CONCEPTS
  - DATA STRUCTURES MORE STABLE THAN FUNCTIONS
  - ONLY SEQUENCING INHERENT IN THE PROBLEM SHOULD CONSTRAIN DESIGN

INSTRUCTOR NOTES

JACKSON STRUCTURED PROGRAMMING OVERVIEW

- WAS DEVELOPED IN THE EARLY 70's
- IS SUITABLE FOR SMALL-MEDIUM PROGRAM DESIGNS
- IS VERY WELL TESTED
- HAS STRUCTURING PRINCIPLES APPLICABLE FOR JSD
- DEVELOPED FOR COBOL SHOPS EXTENDED TO REAL TIME APPLICATIONS
- CONCERNED WITH DESIGNING A PARTICULAR PROGRAM
- A RELIABLE AND REPEATABLE METHOD
- STARTS WITH DATA AND FINISHES WITH A PROGRAM

INSTRUCTOR NOTES

"STRUCTURES" IS THE KEY WORD. SO LET'S LOOK AT THE STRUCTURES JACKSON FEELS ARE CENTRAL TO PROGRAMMING ...

THE WAY YOU WRITE YOUR PROGRAM SHOULD REFLECT THE STRUCTURE OF THE DATA WHICH THE PROGRAM WILL PROCESS.

JACKSON STRUCTURED PROGRAMMING KEY CONCEPT

- PROGRAM STRUCTURES MIRROR

  ⇓

  DATA STRUCTURES WHICH MIRROR

  ⇓

  PROBLEM STRUCTURES

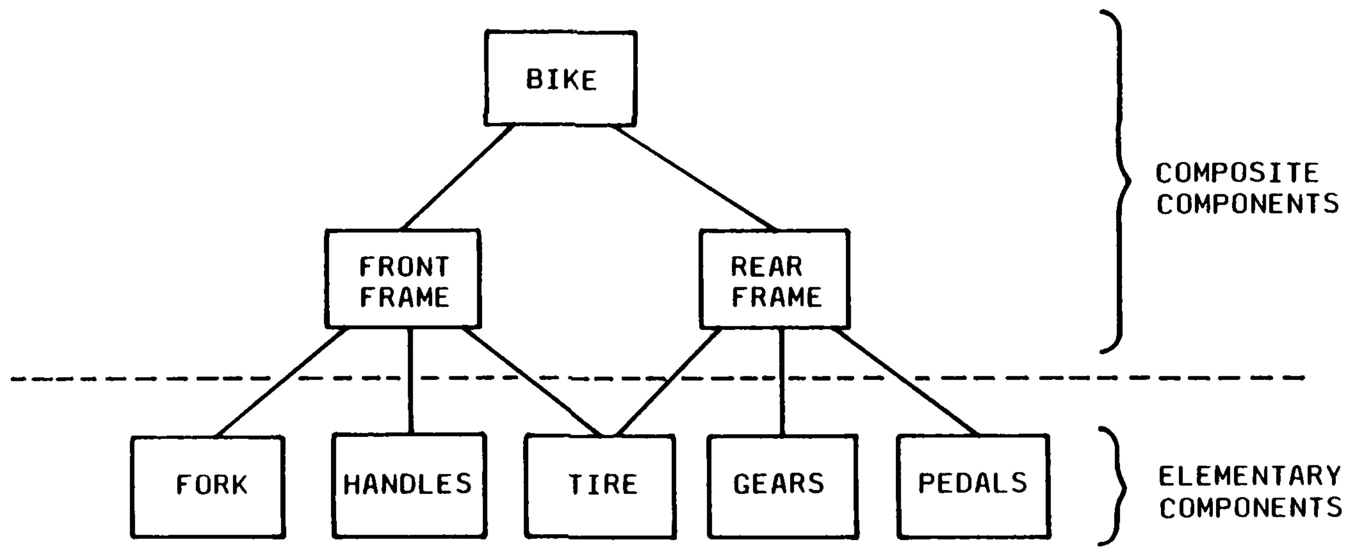
- GRAPHICAL NOTATION IS USED TO EXPRESS PROGRAM
  - COMPONENTS - □
  - SELECTION - ○
  - ITERATION - *
  - HIERARCHY - |

INSTRUCTOR NOTES

ONE MIGHT ARGUE THAT A TIRE CAN BE DECOMPOSED. IT REALLY DEPENDS ON YOUR POINT OF VIEW. IF YOU BUY TIRES FROM A PARTS SUPPLIER, IT IS AN ELEMENTARY COMPONENT.

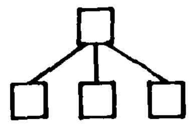
HIERARCHY

- HIERARCHIES ARE STRUCTURES HAVING
  - ELEMENTARY COMPONENTS - THOSE THAT CANNOT BE FURTHER DECOMPOSED
  - COMPOSITE COMPONENTS - THOSE THAT BRING TOGETHER, IN CORRECT RELATIONSHIP, THE PARTS WHICH THEY COMPRISE

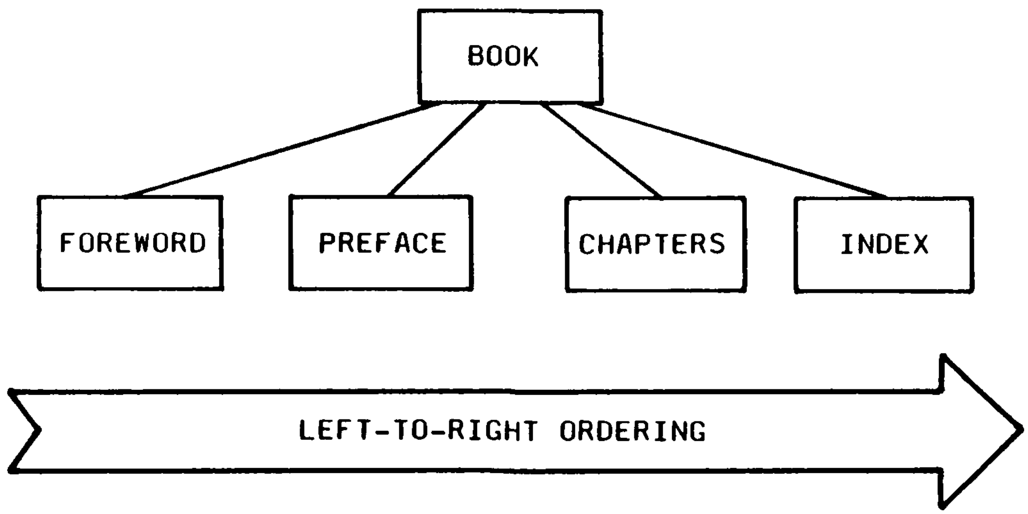


INSTRUCTOR NOTES

THIS CHART ILLUSTRATES THE STRUCTURE TO BE USED AND THEIR REPRESENTATION.

STRUCTURE	REPRESENTATION
SEQUENCE	LEFT-TO-RIGHT READING
SELECTION	
ITERATION	
HIERARCHY	

SEQUENCE

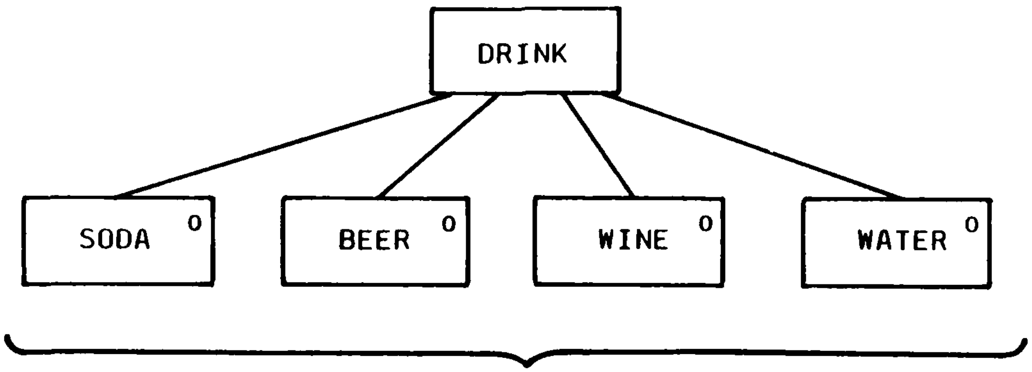


- CAN BE READ AS ...
  - "A BOOK CONSISTS OF FOREWORD, PREFACE, CHAPTERS AND INDEX"

INSTRUCTOR NOTES

POINT OUT THE SYMBOL USED FOR SELECTION.

SELECTION



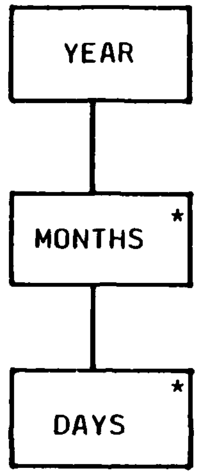
ONE, AND ONLY ONE
OF THESE COMPONENTS
IS SELECTED.

- CAN BE READ AS ...
  - "A DRINK IS EITHER SODA, OR BEER, OR WINE OR WATER"

INSTRUCTOR NOTES

AGAIN, POINT OUT THE ITERATION SYMBOL.

ITERATION



IN GENERAL, THE * MEANS ZERO, ONE OR MANY.

- CAN BE READ AS ...
  - "A YEAR CONTAINS MULTIPLE MONTHS, A MONTH CONTAINS MULTIPLE DAYS"

INSTRUCTOR NOTES

JSP IS A 3-STEP METHOD.

1. FIND THE DATA RELATIONSHIPS THROUGH YOUR OWN UNDERSTANDING OF THE PROBLEM. DRAW THE DATA STRUCTURES.

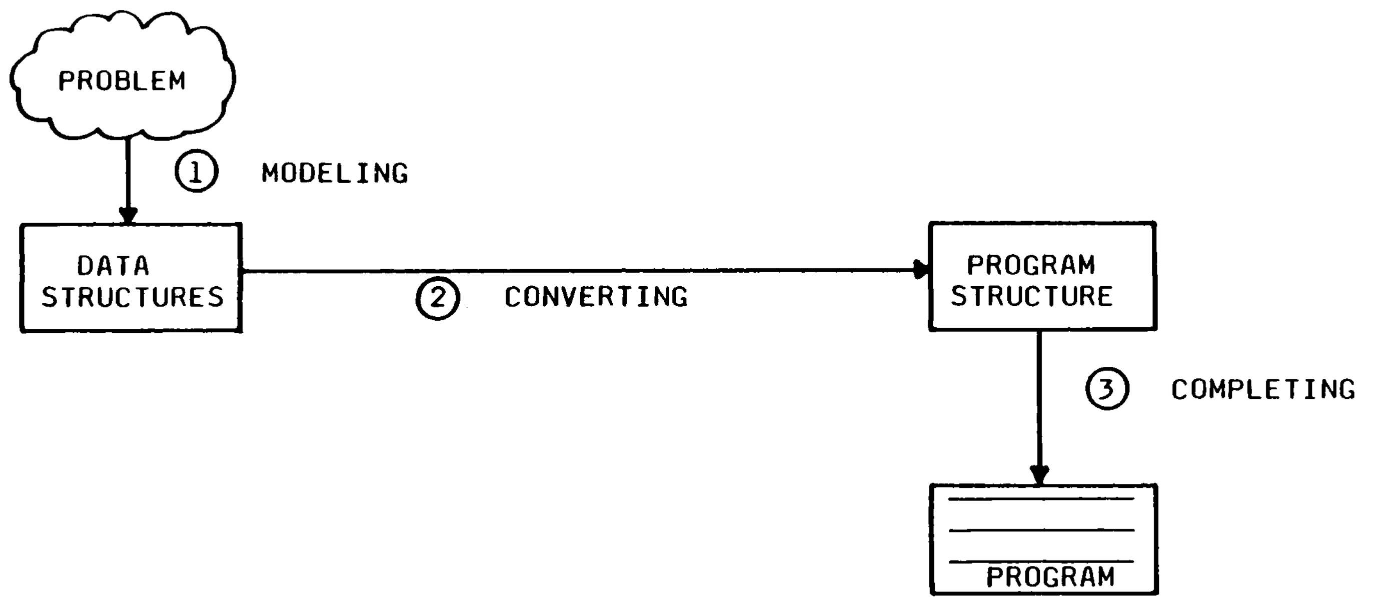
2. CONVERT THESE TO A SKELETON PROGRAM STRUCTURE.

3. FILL IN THE SKELETON.

JACKSON DOESN'T HELP MUCH WITH FINDING THE DATA STRUCTURES. THE KEY IS THAT WE MUST FIND A PROGRAM STRUCTURE THAT MATCHES THE INPUT <u>AND</u> OUTPUT DATA STRUCTURE.

JACKSON STRUCTURED PROGRAMMING METHOD

- THESE STRUCTURES ARE BUILT ACCORDING TO A COOKBOOK-LIKE METHOD ...



<u>MODELING:</u> DEFINE <u>INPUT AND OUTPUT DATA</u> STRUCTURE USING JACKSON'S GRAPHIC NOTATION.

<u>CONVERTING:</u> CREATE A GENERAL PROGRAM STRUCTURE (SAME GRAPHICS) THAT <u>MATCHES</u> THE STRUCTURES OF BOTH THE INPUT AND OUTPUT.

<u>COMPLETING:</u> LIST <u>ELEMENTARY OPERATIONS</u> AND ASSIGN THESE TO MODULES.

INSTRUCTOR NOTES

JACKSON STRUCTURED PROGRAMMING EXAMPLE

<u>GOAL</u>: SUMMARIZE TRUCK ACTIVITY FOR A MILITARY MOTOR POOL.

<u>INPUT</u>: TRUCK FILE, CONTAINING RECORDS (IN TRUCK NUMBER ORDER) THAT TELL ABOUT THE TRUCK'S: SOURCE, DESTINATION, AND LOAD.

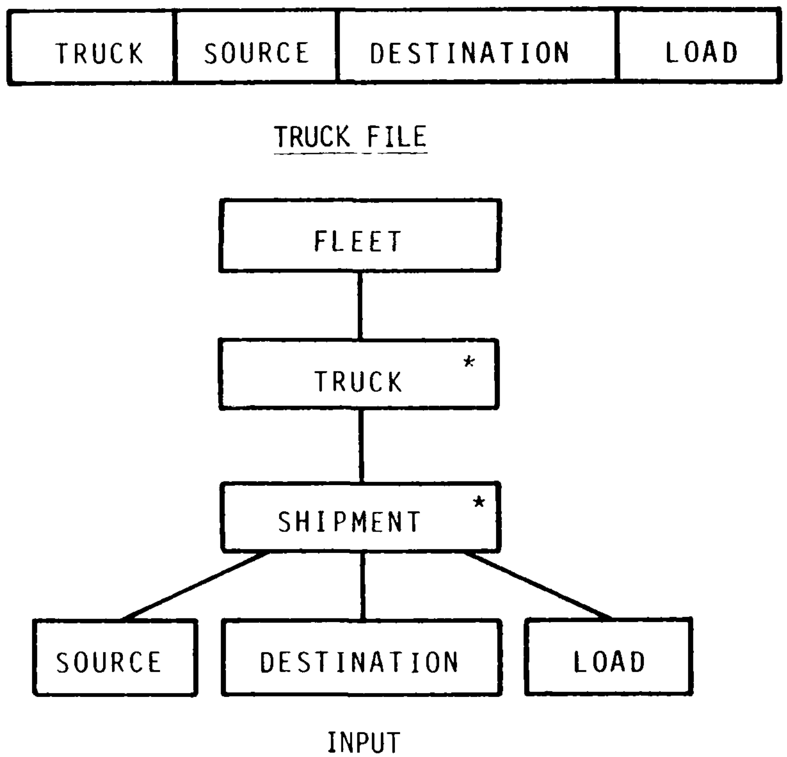
<u>OUTPUT</u>: A REPORT, WITH A LONE TITLE PAGE, SUMMARIZING EACH TRUCK'S SHIPMENT ACTIVITY ON A SINGLE REPORT LINE.

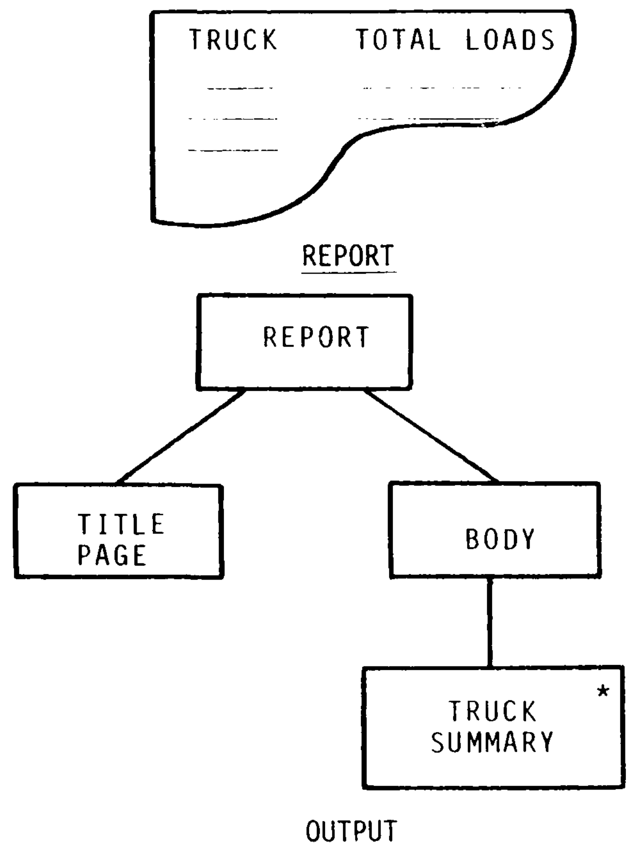
INSTRUCTOR NOTES

MAKE SURE THE CLASS UNDERSTANDS THE DIAGRAMS OR THEY WILL QUICKLY BE LOST.

EXAMPLE

- **MODELING:** DEFINE INPUT AND OUTPUT DATA STRUCTURES

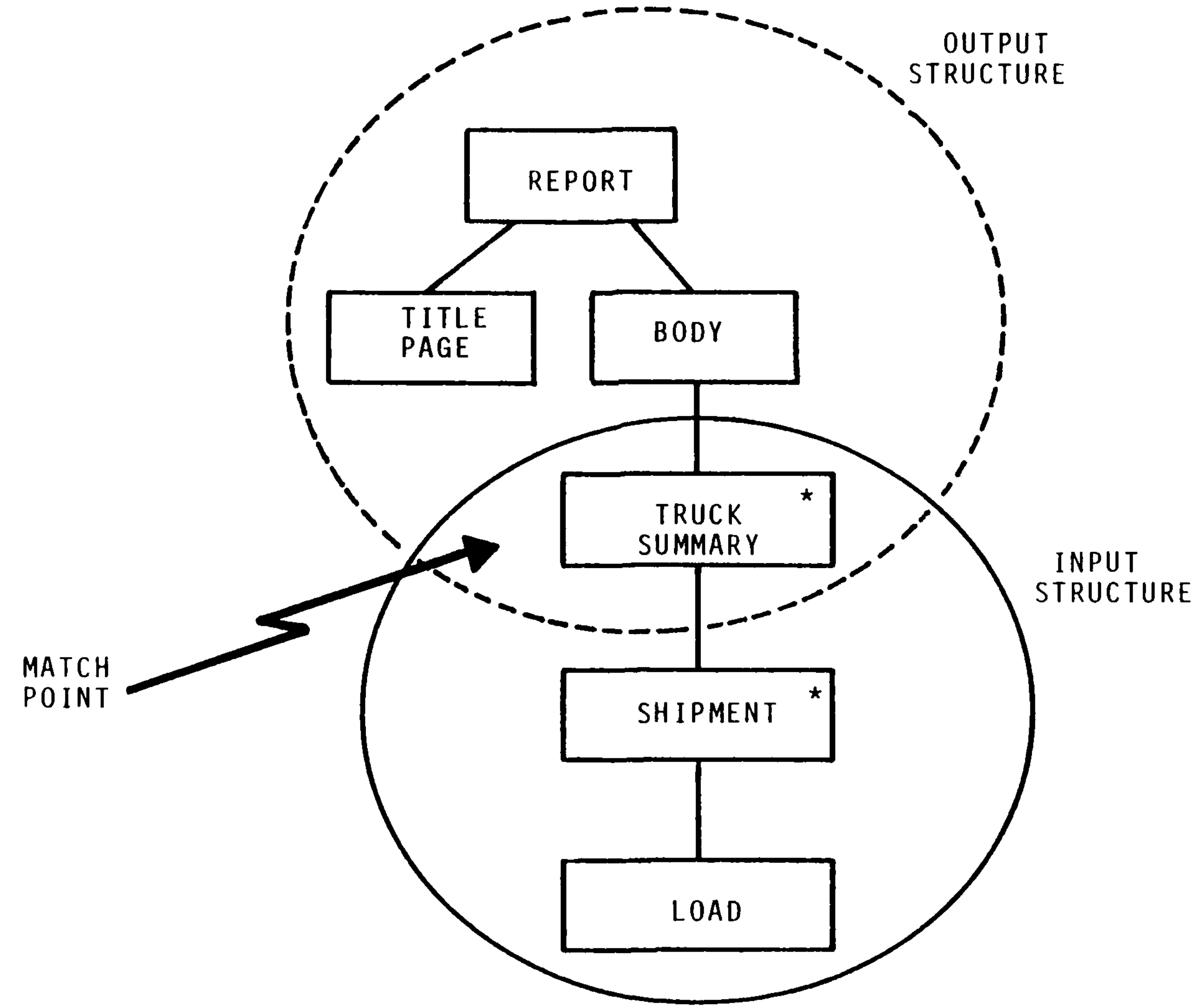




INSTRUCTOR NOTES

EXAMPLE

- <u>CONVERTING</u>: CREATE A GENERAL PROGRAM STRUCTURE THAT MATCHES THE DATA STRUCTURES

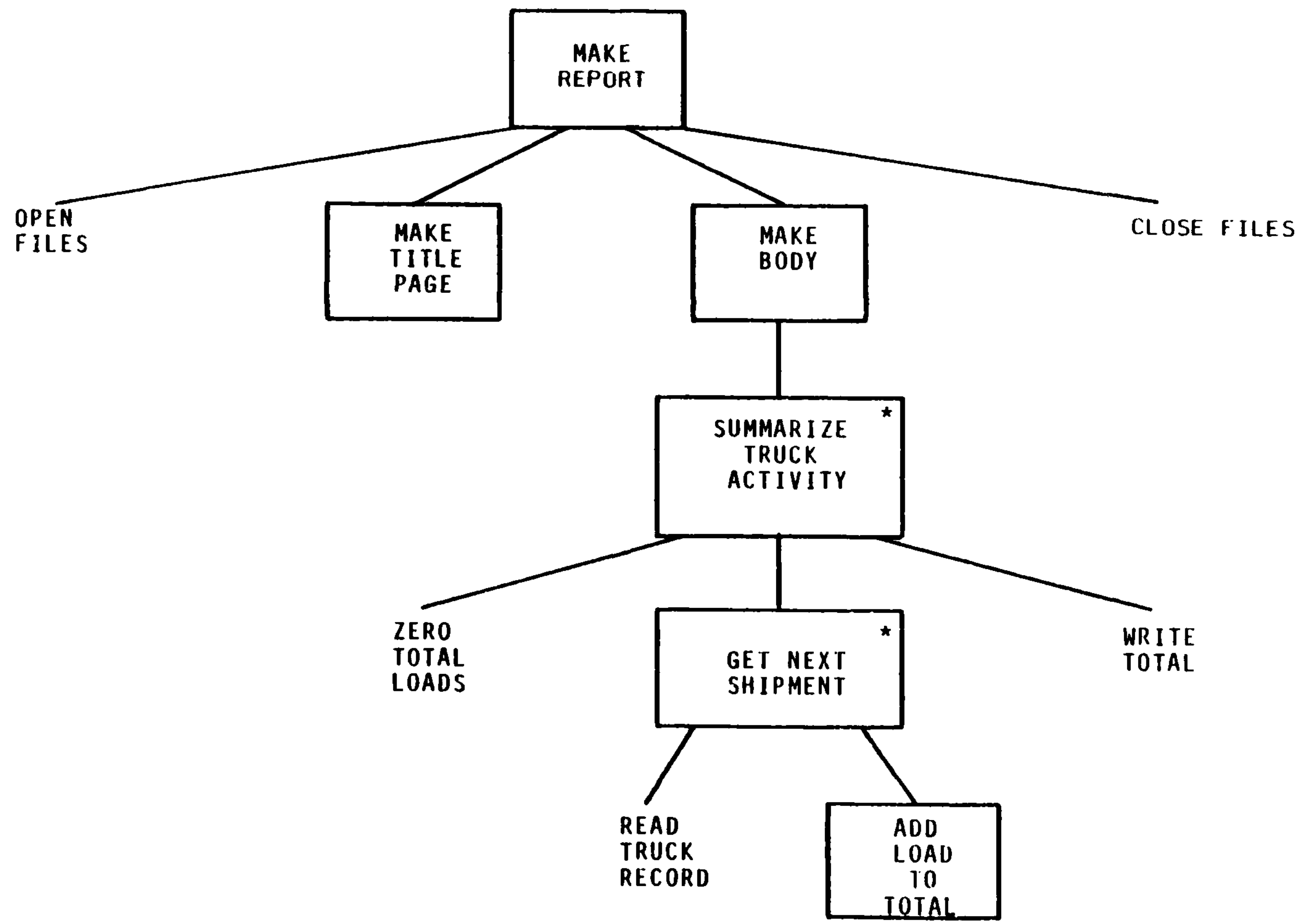


INSTRUCTOR NOTES

THIS IS THE PROCESS: WE ARE NO LONGER ADDRESSING THE DATA STRUCTURE. YET THE STRUCTURE MODELS THE DATA STRUCTURE. THE COMPLETION IS THE ADDITION OF THE FILE MANAGEMENT PROCESSES AND THE INITIALIZATION PHASE.

EXAMPLE

- <u>COMPLETING:</u> LIST ELEMENTARY OPERATIONS AND ASSIGN THEM TO MODULES



INSTRUCTOR NOTES

LIMITATIONS

- NOT ALL PROBLEMS ARE SO SIMPLE ...
  - PROGRAMS MUST MAKE PARTIAL DECISIONS BEFORE HAVING A COMPLETE SET OF DATA (SOLUTION USE BACKTRACKING)
  - THE STRUCTURE OF INPUT AND OUTPUT DATA DO NOT MATCH (SOLUTION RESOLVE STRUCTURE CLASH)
- LARGE PROGRAMS HAVE A VERY LARGE NUMBER OF DATA STRUCTURES MAKING METHODS HARD TO HANDLE

INSTRUCTOR NOTES

ASK WHEN WOULD "INCOMPLETE DATA" OCCUR?

BACKTRACKING

- INCOMPLETE DATA REQUIRES THE DESIGN TECHNIQUE OF <u>BACKTRACKING</u>:

  - MAKE AN ASSUMPTION ABOUT WHAT IS COMING NEXT
  - CONTINUE PROCESSING UNTIL YOU FIND OUT YOUR ASSUMPTION WAS WRONG
  - UNDO ANY DAMAGE AND MAKE A NEW ASSUMPTION BASED ON THE ADDITIONAL INFORMATION

INSTRUCTOR NOTES

BACKTRACKING EXAMPLE

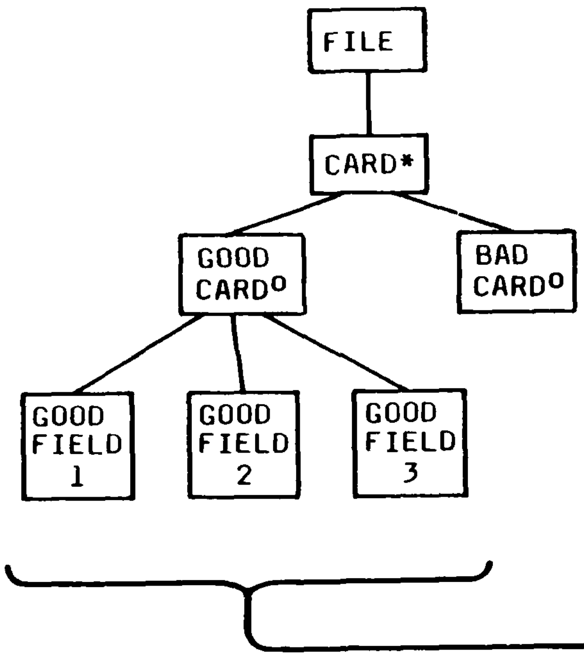
- WE <u>ASSUME</u> THE CARD IS GOOD UNTIL PROVEN WRONG. WE "BACK OUT" OF OUR ASSUMPTION WHEN A FIELD IS FOUND TO BE BAD:

```
Process_Good_Card start
   Process_Field_1
   quit if bad Field_1
   Process_Field_2
   quit if bad Field_2
   Process_Field_3
   quit if bad Field_3
   Process_Whole_Card (admit that card is good)
Process_Good_Card end
```

INSTRUCTOR NOTES

BACKTRACKING

EXAMPLE



FILE

WE ARE GIVEN A FILE OF CARDS. EACH CARD IS EITHER GOOD OR BAD. GOOD CARDS HAVE GOOD FIELDS; BAD CARDS HAVE AT LEAST ONE BAD FIELD.

BUT WE DON'T KNOW A FIELD IS BAD UNTIL WE PROCESS IT!

INSTRUCTOR NOTES

THE THIRD IMPLEMENTATION WAY IS NOT CONVENIENT IN THE COBOL WORLD, BUT IS USEFUL IN THE Ada WORLD.

STRUCTURE CLASH

- PROBLEM: SOMETIMES NO MATCH CAN BE FOUND BETWEEN INPUT AND OUTPUT DATA STRUCTURES. (A SINGLE PROGRAM CANNOT BE DEFINED FROM JUST THESE STRUCTURES.)

- BASIC APPROACH: CREATE INTERMEDIATE DATA STRUCTURES THAT REORGANIZE, IN POSSIBLY SMALL STEPS, THE DATA UNTIL A MATCH IS ACHIEVED.

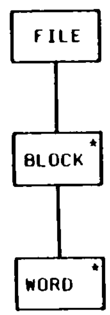
- THERE ARE THREE WAYS TO IMPLEMENT THE APPROACH:
  - CREATE AN <u>INTERMEDIATE FILE</u>
  - CREATE TEMPORARY IN-CORE DATA STRUCTURES (<u>PROGRAM INVERSION</u>)
  - USE ASYNCHRONOUS TASKS, COUPLED WITH BUFFERS

INSTRUCTOR NOTES

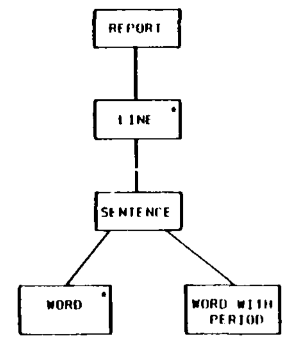
NOTICE THAT ONLY AT "WORD*" IS THERE A MATCH.

STRUCTURE CLASH

INPUT: TEXT IS STORED IN BLOCKS OF WORDS.

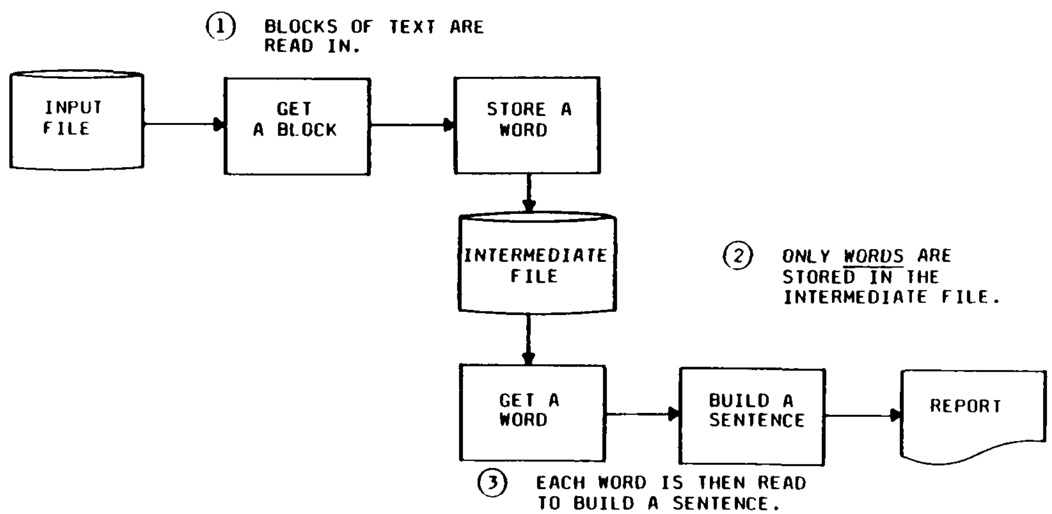


OUTPUT: THE REPORT MUST WRITE ON ONE LINE.



← NO MATCH POINT →

INTERMEDIATE FILE APPROACH



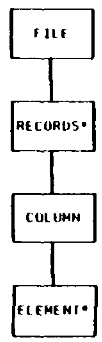
INSTRUCTOR NOTES

POINT OUT THAT THE OUTPUT PROGRAM IS "INVERTED" FROM THE OUTPUT FORMAT.

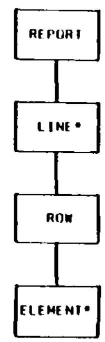
THE IN-CORE DATA STRUCTURE HOLDS ALL THE DATA READ IN BEFORE THE DATA IS WRITTEN OUT.

STRUCTURE CLASH

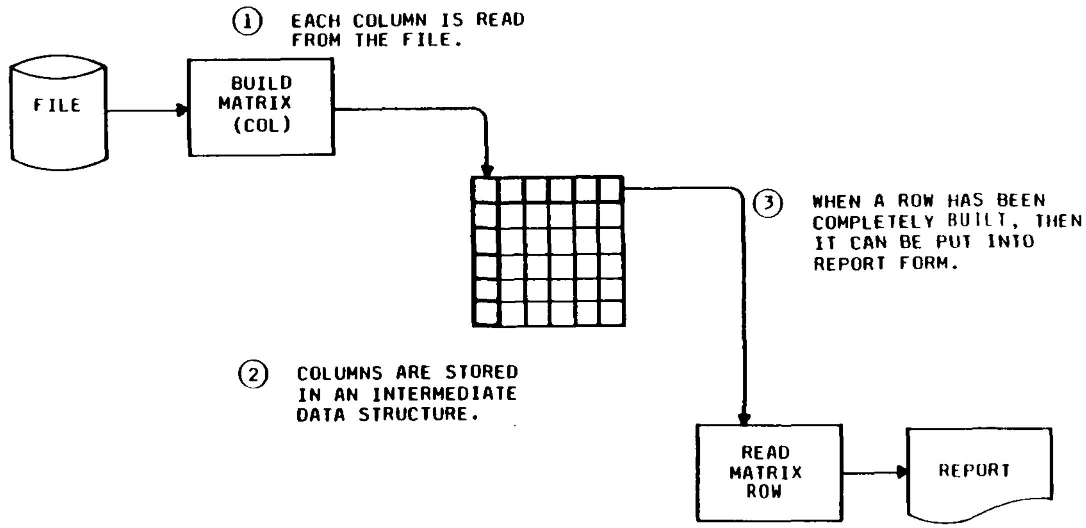
INPUT: A MATRIX IS STORED IN COLUMN-MAJOR ORDER.



OUTPUT: A REPORT MUST BE READ IN ROW-MAJOR ORDER.



PROGRAM INVERSION APPROACH



INSTRUCTOR NOTES

MAKE THE COMPARISON IN NOTATION USED IN STRUCTURED DESIGN AND JACKSON METHOD.

JACKSON STRUCTURED PROGRAMMING SUMMARY

- USES GRAPHICS TO REPRESENT DATA AND ALGORITHM STRUCTURES CONSISTENTLY.
- IS A REPEATABLE METHOD (OF MODELING, CONVERTING AND COMPLETING) THAT FORMS A PROGRAM.
- RELIES ON MATCHING DATA STRUCTURES.
- USES INTERMEDIATE FILES AND PROGRAM INVERSION TO RESOLVE STRUCTURE CLASHES.

INSTRUCTOR NOTES

THEME: SAME AS JACKSON

PURPOSE: SAME AS JACKSON

REFERENCES:

WIRTH, N. *"LOGICAL CONSTRUCTION OF PROGRAMS" LEIDEN, NETHERLANDS;* 1974

ORR, K. "STRUCTURED SYSTEMS DEVELOPMENT" YOURDON, NYC; 1977

Section 18

WARNIER-ORR METHOD

INSTRUCTOR NOTES

WARNIER-ORR IS CONSIDERED TO BE VERY SIMILAR TO JACKSON. WARNIER AND JACKSON DEVELOPED THEIR IDEAS AROUND THE SAME TIME. ORR EXTENDED THESE AND POPULARIZED THEM. IT USES A DIFFERENT FORM OF GRAPHICS BUT USES JACKSON COMPATIBLE GRAPHICS FOR DATA AND FUNCTIONS.

OVERVIEW

- THE WARNIER-ORR METHOD ...
  - WAS DEVELOPED IN FRANCE BY JEAN WARNIER
  - WAS MODIFIED AND PUBLICIZED BY KEN ORR
  - USES GRAPHICS TO MODEL BOTH DATA AND FUNCTIONS
  - DEPARTS FROM THE BOX AND ARROW NOTATION

INSTRUCTOR NOTES

THERE ARE THREE (3) ELEMENTS TO THE WARNIER SYNTAX ...

NAMES ARE EITHER

- NOUNS -- FOR DATA
- *VERBS -- FOR FUNCTIONS*

THERE ARE THREE (3) KINDS OF GRAPHIC SYMBOLS THAT CONNECT (I.E. STRUCTURE) NAMES ...

BRACES AND LOGIC SYMBOLS DEFINE RELATIONSHIPS, WHEREAS PARENTHESES DEFINE ATTRIBUTES.

WARNIER DIAGRAM

SYNTAX

WARNIER DIAGRAM SYNTAX	NAMES	NOUNS VERBS
	CONNECTIVES	BRACE PARENTHESES LOGIC SYMBOL
	ORDER	

VG 778.1

INSTRUCTOR NOTES

HERE IS AN EXAMPLE OF THE SEVEN FUNCTIONS NEEDED TO HANDLE A QUERY. LEFT OF BRACE IS PARENT FUNCTION. RIGHT OF BRACE IS DECOMPOSITION ...

CONNECTIVES

SYNTAX

- BRACES SHOW HIERARCHICAL DECOMPOSITION:

HANDLE QUERY	GET A VALID REQUEST	READ A REQUEST
		VALIDATE A REQUEST
	EXTRACT FROM DATABASE	
	WRITE FORMATTED REPLY	FORMAT RESPONSE
		WRITE FORMATTED RESPONSE

INSTRUCTOR NOTES

HERE IS A DESCRIPTION OF A COMPANY'S SALES STAFF. THERE ARE EIGHT REGIONS, FIVE DISTRICTS, A VARIABLE NUMBER OF SALES PEOPLE EACH RESPONSIBLE FOR A VARIABLE NUMBER OF CUSTOMERS. ALSO INCLUDED IN THE DESCRIPTION ARE DETAILS ABOUT EACH CUSTOMER.

CONNECTIVES

SYNTAX

- PARENTHESES INDICATE <u>REPETITION</u>:

COMPANY { REGION (8) { DISTRICT (5) { SALESPERSON (S) { CUSTOMER (C) { NAME, ADDRESS, SALES, PROFIT

  - NUMBER OF REPETITIONS

INSTRUCTOR NOTES

HERE, AN ARITHMETIC OPERATION IS CONSIDERED TO BE EITHER: ADDITION, SUBTRACTION, MULTIPLICATION, OR DIVISION.

CONNECTIVES

SYNTAX

- THE LOGICAL SYMBOL $\oplus$ INDICATES ALTERNATIVES.

$$
\text{DO ARITHMETIC OPERATION} \left\{ \begin{array}{c} \text{DO ADDITION OPERATION} \\ \oplus \\ \text{DO SUBTRACTION OPERATION} \\ \oplus \\ \text{DO MULTIPLICATION OPERATION} \\ \oplus \\ \text{DO DIVISION OPERATION} \end{array} \right.
$$

RULE: FOR ANY INSTANCE, ONE, AND ONLY ONE, OF THE ALTERNATIVES IS VALID.

INSTRUCTOR NOTES

MATCHING INPUT TO OUTPUT IS NOT CONSIDERED IN THIS METHODOLOGY.

DESIGN STRATEGIES

- WARNIER-ORR, LIKE THE JACKSON METHOD, CONCENTRATES ON DATA <u>BEFORE</u> PROGRAMS.

- DEVELOP A PROGRAM BY:

  - FIRST UNDERSTANDING ITS <u>INPUT</u>, OR

  - FIRST UNDERSTANDING ITS <u>OUTPUT</u>.

- BUT ALWAYS UNDERSTAND ITS <u>DATA</u> FIRST BY:

  - DRAWING TABLES TO UNDERSTAND WHEN INPUTS CAUSE FUNCTIONS TO BE PERFORMED

INSTRUCTOR NOTES

THIS TABLE STATES THAT ...

FUNCTION	IS DONE WHEN	DATA
W		A IS NOT PRESENT
X		B IS PRESENT
Y		B OR C IS PRESENT
Z		C IS PRESENT

DESIGN STRATEGIES
EXAMPLE

- TABLES TAKE ON THE FORM OF A DECISION TABLE

DATA	FIELD A	0	0	0	0	1	1	1	1
	FIELD B	0	0	1	1	0	0	1	1
	FIELD C	0	1	0	1	0	1	0	1
FUNCTION	W	X	X	X	X				
	X			X	X			X	X
	Y		X	X	X		X	X	X
	Z		X		X		X		X

  - A 1 INDICATES WHETHER FIELD IS PRESENT; A 0 INDICATES FIELD IS ABSENT
  - LETTER INDICATES DO FUNCTION NAMED BY THAT LETTER

INSTRUCTOR NOTES

THIS COULD BE THE RESULTING DESIGN IF FIELDS B AND C <u>NEVER</u> ARE PRESENT AT THE SAME TIME.

DESIGN STRATEGIES

EXAMPLE

- WARNIER-ORR DIAGRAM

PROGRAM
- OPEN FILES
- PROCESS EACH RECORD (N)
  - FIELD A ABSENT
    - DO FUNCTION W
  - FIELD B PRESENT
    - DO FUNCTION X
    - DO FUNCTION Y
  - FIELD C PRESENT
    - DO FUNCTION Y
    - DO FUNCTION Z
- CLOSE FILES

INSTRUCTOR NOTES

A WARNIER DIAGRAM IS READ LEFT-TO-RIGHT AND TOP-TO-BOTTOM.

WARNIER METHOD

(LOGICAL CONSTRUCTION OF PROGRAMS)

- THE PROCEDURE
  - IDENTIFY ALL INPUT DATA ITEMS
  - ORGANIZE THE INPUT DATA INTO A HIERARCHY (USE DIAGRAM FORMAT)
  - LABEL DIAGRAM WITH LOGIC SYMBOLS AND REPETITION COUNTS
  - REPEAT PROCEDURE FOR OUTPUT DATA ITEMS
  - SPECIFY FUNCTIONAL NATURE OF PROGRAM BY DEFINING TYPE OF OPERATIONS PERFORMED IN THIS ORDER
    - READ INPUT OPERATIONS
    - BRANCH OPERATIONS
    - COMPUTATIONS
    - OUTPUTS
    - CALLS TO SUBROUTINES
  - REFINE THE ABOVE FUNCTIONAL DIAGRAM

INSTRUCTOR NOTES

LET'S LOOK AGAIN AT THE PARTS MOVEMENT PROBLEM ...

PARTS - MOVEMENT PROBLEM

- <u>INPUT</u>:
  - FILE OF RECORDS, SORTED BY PART NUMBER,
  - EACH RECORD CONTAINS INFORMATION ON ONE ISSUE OR RECEIPT OF ONE ITEM.
- <u>REQUIRED OUTPUT</u>:
  - REPORT SHOWING NET MOVEMENT OF EACH ITEM
  - HEADING ON REPORT

INSTRUCTOR NOTES

FOR INPUT, THERE ARE "M" MOVEMENT RECORDS FOR EACH PART, AND THERE ARE "N" PARTS IN THE FILE.

FOR OUTPUT, THE REPORT COMPRISES A HEADING AND "X" OUTPUT LINES, EACH GIVING THE TOTAL MOVEMENT FOR A PART.

PARTS - MOVEMENT PROBLEM

- <u>INPUT</u>:

$$\text{FILE} \left\{ \begin{array}{c} \text{PART} \\ (N) \end{array} \left\{ \begin{array}{c} \text{MOVEMENT} \\ \text{RECORD} \\ (M) \end{array} \left\{ \begin{array}{c} \text{ISSUE} \\ \oplus \\ \text{RECEIPT} \end{array} \right. \right. \right.$$

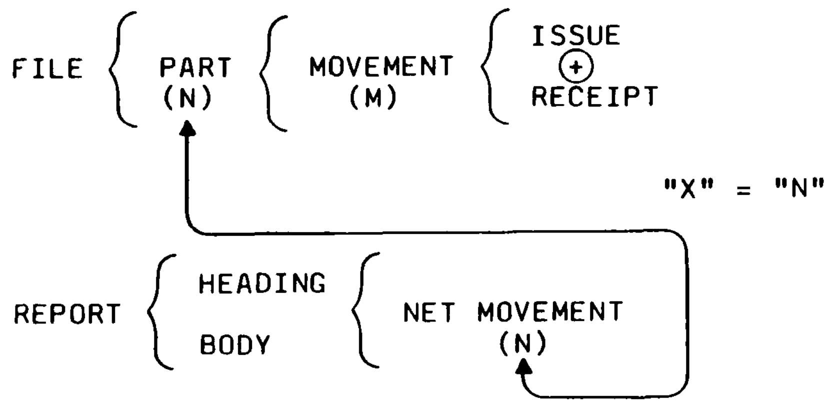
- <u>OUTPUT</u>:

$$\text{REPORT} \left\{ \begin{array}{l} \text{HEADING} \\ \\ \text{BODY} \end{array} \left\{ \begin{array}{c} \text{NET MOVEMENT LINE} \\ (X) \end{array} \right. \right.$$

INSTRUCTOR NOTES

IN THIS CASE, THE NUMBER OF NET MOVEMENT LINES ("X") MUST EQUAL THE NUMBER OF PARTS IN THE FILE ("N"). SO "X" = "N" AND THE STRUCTURES CAN BE MATCHED ...

PARTS - MOVEMENT PROBLEM



MATCHING THE REPETITIONS OF OUTPUTS TO INPUTS

INSTRUCTOR NOTES

HERE'S THE MERGED STRUCTURES. THESE GIVE AN OVERALL STRUCTURE OF THE PROGRAM.

PARTS - MOVEMENT PROBLEM

```
MAKE REPORT
    MAKE HEADING
    MAKE BODY
        MAKE NET MOVEMENT LINE (N)
            PROCESS MOVEMENT RECORD (M)
                PROCESS ISSUE
                ⊕
                PROCESS RECEIPT
```

INSTRUCTOR NOTES

AND HERE'S THE "COMPLETED" PROGRAM, WITH ADDITIONS TO THE BASIC STRUCTURE PERTAINING TO FILES AND VARIABLES ...

PARTS - MOVEMENT PROBLEM

- MAKE REPORT
  - OPEN FILES
  - MAKE HEADING
  - MAKE BODY
    - MAKE NET MOVEMENT LINE (N)
      - ZERO NET MOVEMENT
      - PROCESS MOVEMENT RECORD (M)
        - READ NEXT RECORD
        - UPDATE NEXT MOVEMENT
          - SUBTRACT FROM NET MOVEMENT
          - ⊕
          - ADD TO NET MOVEMENT
      - WRITE NET LINE
  - CLOSE FILES

INSTRUCTOR NOTES

COMPLETING THE DESIGN

- WARNIER-ORR METHOD CAN BE VIEWED AS A SUBSET OF JACKSON STRUCTURED PROGRAMMING USING A DIFFERENT NOTATION

- MUST STILL USE JACKSON CONCEPTS TO DEVELOP FINAL PROGRAM STRUCTURE:

  - MATCHING STRUCTURES

  - RESOLVING CLASHES

  - COMPLETING THE PROGRAM

INSTRUCTOR NOTES

THEME: HOS IS A METHOD FOR DEVELOPING PROVABLY CORRECT SOFTWARE SYSTEMS

PURPOSE: PROVIDE AN OVERVIEW OF THE DESIGN METHODS ASSOCIATED WITH HOS

REFERENCES:

HAMILTON, M., ZELDIN, S. "HIGHER-ORDER SOFTWARE: A METHODOLOGY FOR DEFINING SOFTWARE" CHARLES STARK DRAPER LABS, MA; R-862; MARCH 1975

MARTIN, J. "SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS" PRENTICE-HALL INC., NJ; 1985

Section 19

HIGHER ORDER SOFTWARE METHOD

INSTRUCTOR NOTES

- HOS HELPS MODEL SYNCHRONOUS, ASYNCHRONOUS, NETWORKED, REAL TIME, INTERRUPT-DRIVEN, RECURSIVE, AND INTERACTIVE SYSTEMS.
- IT HAS BEEN AROUND IN SOME FORM SINCE THE MID-60's, AND HAS BEEN USED BY NASA SINCE THE APOLLO SPACE PROGRAM. EXTENSIVE USE IN SHUTTLE PROGRAM.
- IT IS VERY HIERARCHICAL IN NATURE.

OVERVIEW

- HIGH ORDER SOFTWARE ...
  - GREW FROM THE APOLLO MOON-MISSION EFFORT
  - DEALS WITH SOFTWARE SYSTEMS
  - WAS DEVELOPED FOR REAL TIME SYSTEMS
  - EMPHASIS IS ON GENERATION OF PROVABLY CORRECT SOFTWARE
  - SIMPLIFIES WRITING SOFTWARE THAT IMPLEMENTS MATHEMATICAL EQUATIONS
  - HAS BEEN USED AS AN ANALYSIS AS WELL AS DESIGN METHOD

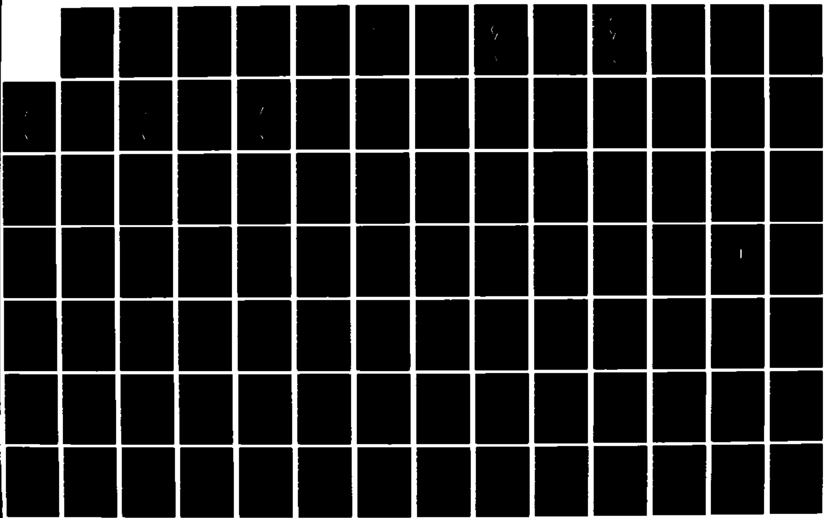
INSTRUCTOR NOTES

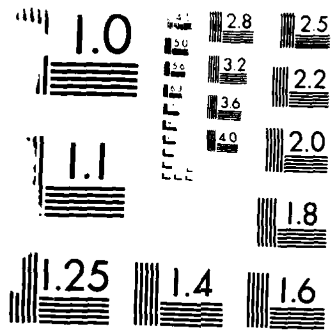
KEYWORDS OF HOS ARE ...

- SEQUENCING -- PLUGGING TOGETHER FUNCTIONS INTO A WORKING <u>THREAD</u>.
- FUNCTIONS -- THE ATOMIC PARTS OF THE SOFTWARE SYSTEM.

AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE 5/
ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2
(U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

UNCLASSIFIED F/G 5/9 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

HIGHER ORDER SOFTWARE

FUNDAMENTAL BASIS

- SOFTWARE SYSTEMS CAN BE REPRESENTED AS A HIERARCHICAL SYSTEM MODEL DEVELOPED IN ACCORDANCE WITH SIX AXIOMS

- BASIC NODE SYNTAX

```
OUTPUT OBJECTS +----------+ INPUT OBJECTS
-------------- | FUNCTION | -------------
               +----------+
```

- OBJECTS CAN BE ...
  - DATA ITEM
  - LIST
  - TABLE
  - REPORT
  - PHYSICAL ENTITY

- A FUNCTION CAN BE ...
  - MATHEMATICAL FUNCTION
  - PROGRAMMED ALGORITHM
  - DATABASE QUERY
  - A BROAD STATEMENT OF REQUIREMENTS

INSTRUCTOR NOTES

(a) AXIOMS CONTROL DECOMPOSITIONS. PARENTS CAN INVOKE ONLY ITS OFFSPRING, CONTROLS THE ORDER OF THEIR INVOCATION, AND CONTROLS THEIR INPUT AND OUTPUT (i.e., ACCESS RIGHTS).

(b) ALL CONTROL STRUCTURES ARE DERIVED FROM THE AXIOMS.

(c) DATA TYPES AUTOMATICALLY DEFINE PRIMITIVE OPERATIONS ON DATA.

(d) FUNCTIONS REPRESENT SPECIFIC TRANSFORMATIONS OF INPUT VALUES OF DATA TYPE MEMBERS TO OUTPUT VALUES OF DATA TYPE MEMBERS.

(e) THE AXIOMATIC NATURE OF HOS ALLOWS THE OUTPUTS TO BE TRANSFORMED INTO REPRESENTATIONS TO BE USED AS INPUTS TO JACKSON, PSL/PSA, SADT, SREM, AND PETRI NETS.

HIGHER ORDER SOFTWARE
FUNDAMENTAL BASIS

- HIERARCHY REPRESENTS DECOMPOSITION OF SOFTWARE FROM HIGHEST LEVEL (TOP) TO PRIMITIVE OR PREVIOUSLY DEVELOPED FUNCTIONS.

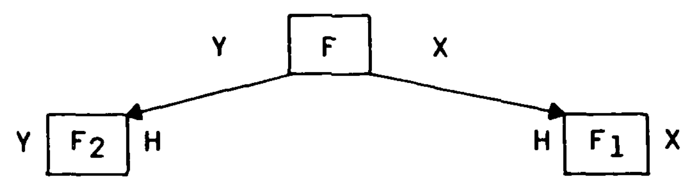
- THE SIX AXIOMS THAT MAKE FUNCTION DECOMPOSITION MATHEMATICALLY PRECISE.
  - AXIOM 1: A GIVEN MODULE CONTROLS THE INVOCATION OF THE SET OF FUNCTIONS ON ITS IMMEDIATE, AND ONLY ITS IMMEDIATE, LOWER LEVEL.
  - AXIOM 2: A GIVEN MODULE CONTROLS THE RESPONSIBILITY FOR ELEMENTS OF ONLY ITS OWN OUTPUT SPACE.
  - AXIOM 3: A GIVEN MODULE CONTROLS THE OUTPUT ACCESS RIGHTS TO EACH SET OF VARIABLES WHOSE VALUES DEFINE THE ELEMENTS OF THE OUTPUT SPACE FOR EACH IMMEDIATE, AND ONLY EACH IMMEDIATE, LOWER LEVEL FUNCTION.
  - AXIOM 4: A GIVEN MODULE CONTROLS THE INPUT ACCESS RIGHTS TO EACH SET OF VARIABLES WHOSE VALUES DEFINE THE ELEMENTS OF THE INPUT SPACE FOR EACH IMMEDIATE, AND ONLY EACH IMMEDIATE, LOWER LEVEL FUNCTION.
  - AXIOM 5: A GIVEN MODULE CONTROLS THE REJECTION OF INVALID ELEMENTS OF ITS OWN, AND ONLY ITS OWN, INPUT SET.
  - AXIOM 6: A GIVEN MODULE CONTROLS THE ORDERING OF EACH TREE FOR THE IMMEDIATE, AND ONLY THE IMMEDIATE, LOWER LEVEL.

- HIERARCHY CAN BE USED WITH PROGRAM GENERATOR TO "AUTOMATICALLY" PRODUCE "CORRECT" PROGRAMS.

INSTRUCTOR NOTES

FOR JOIN, ONE OFFSPRING DEPENDS ON ANOTHER.

FOR EXAMPLE:

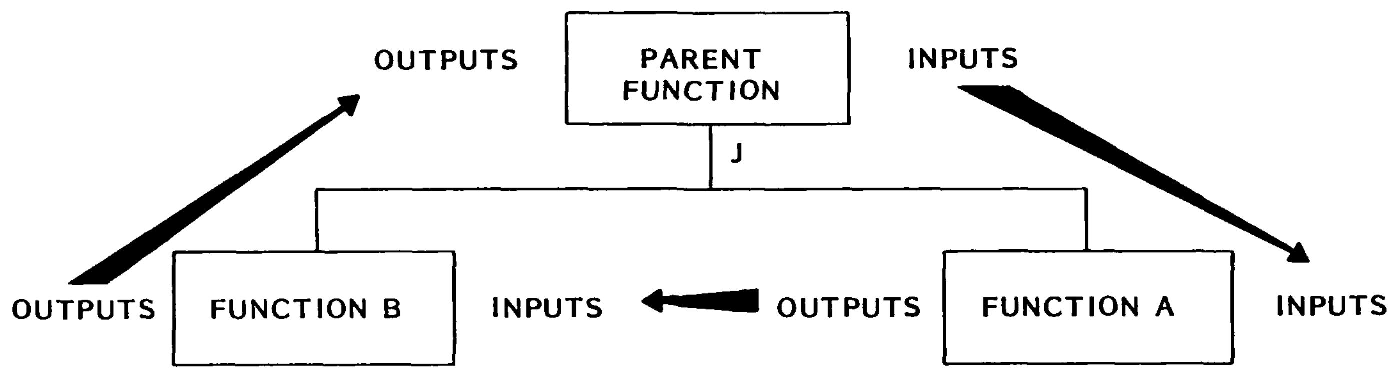


WHERE $Y = F(X)$, FIRST $H = F_1(X)$ THEN $Y = F_2(H)$

PRIMITIVE CONTROL STRUCTURE

(JOIN)

JOIN,J



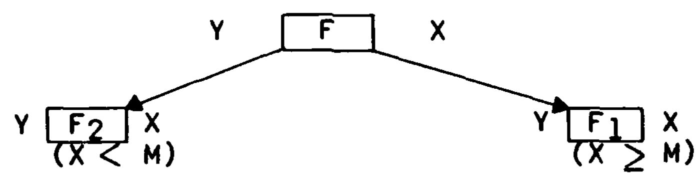
FUNCTION A IS PERFORMED FIRST, THEN FUNCTION B

- INPUTS TO RIGHT OFFSPRING ARE IDENTICAL TO INPUTS TO PARENT (INCLUDING ORDER).
- INPUTS TO LEFT OFFSPRING ARE IDENTICAL TO OUTPUTS FROM RIGHT OFFSPRING (INCLUDING ORDER).
- OUTPUTS FROM PARENT ARE IDENTICAL TO OUTPUTS FROM LEFT OFFSPRING (INCLUDING ORDER).

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

FOR "OR", WE HAVE



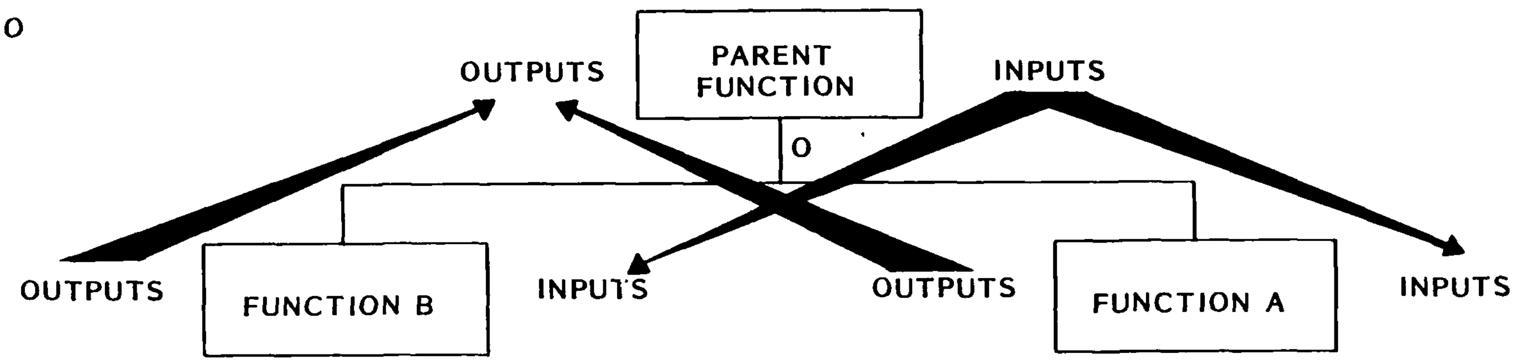
FIRST, X IS CHECKED TO SEE IF IT IS $\geq$ M (IF IT IS $Y = F_1(X)$).

IF IT IS NOT, $Y = F_2(X)$.

PRIMITIVE CONTROL STRUCTURE

(OR)

OR,O



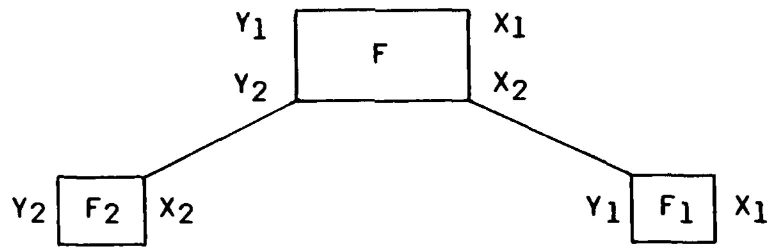
FUNCTION A OR FUNCTION B IS PERFORMED

- INPUTS TO ALL FUNCTIONS ARE IDENTICAL (INCLUDING ORDER).
  - ALL INPUTS ARE USED IN A PARTITION FUNCTION P WHICH CHOOSES WHICH OFFSPRING TO EXECUTE.
- OUTPUTS FROM ALL FUNCTIONS ARE IDENTICAL (INCLUDING ORDER).

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

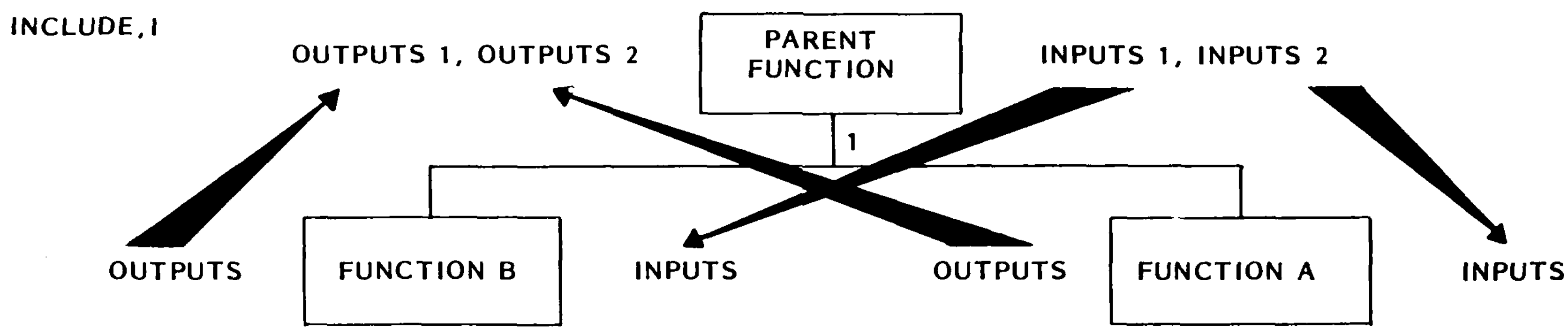
INSTRUCTOR NOTES

USING INCLUDE, EACH OFFSPRING PRODUCES PART OF THE OUTPUT



PRIMITIVE CONTROL STRUCTURE

(INCLUDE)

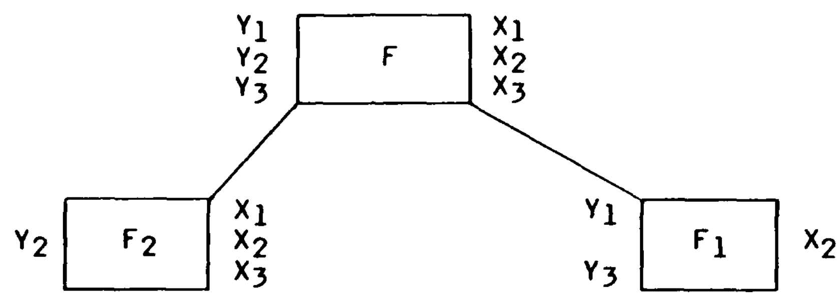


FUNCTION A AND FUNCTION B ARE BOTH PERFORMED

- INPUTS TO LEFT OFFSPRING ARE IDENTICAL TO FIRST INPUTS TO PARENT (INCLUDING ORDER).
- INPUTS TO RIGHT OFFSPRING ARE IDENTICAL TO REMAINING INPUTS TO PARENT (INCLUDING ORDER).
  - INPUTS OF OFFSPRING ARE EXCLUSIVE OF EACH OTHER AND INCLUDE ALL THE PARENT'S INPUTS.
- FIRST OUTPUTS FROM PARENT ARE IDENTICAL TO OUTPUTS FROM LEFT OFFSPRING (INCLUDING ORDER).
- REMAINING OUTPUTS FROM PARENT ARE IDENTICAL TO OUTPUTS FROM RIGHT OFFSPRING (INCLUDING ORDER).
  - OUTPUTS OF OFFSPRING ARE EXCLUSIVE OF EACH OTHER AND INCLUDE ALL PARENT'S OUTPUTS.

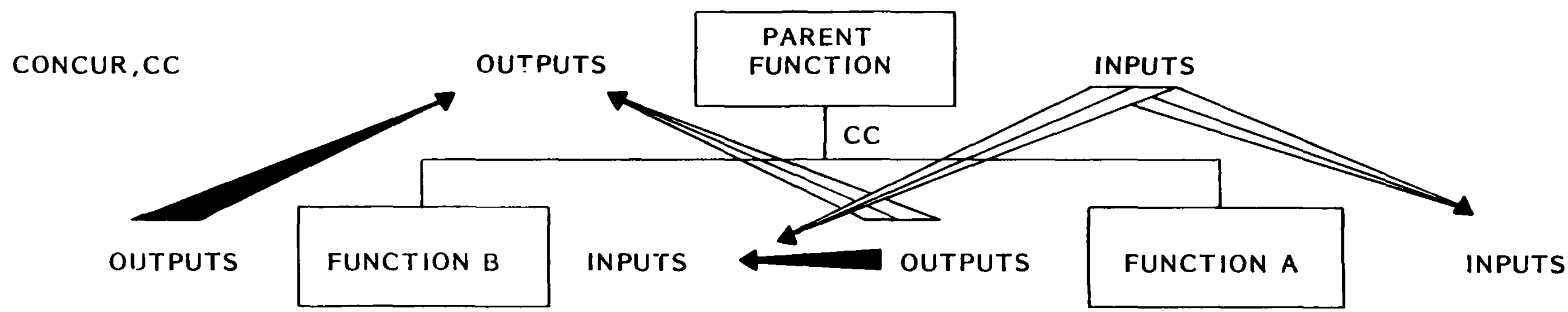
SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

CONCUR USES THE SAME INPUT IN MULTIPLE PLACES.



NON-PRIMITIVE CO CONTROL STRUCTURE

(CONCUR)



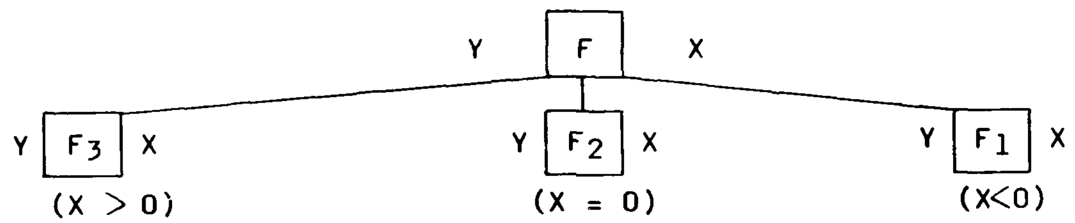
FUNCTION A IS PERFORMED FIRST, THEN FUNCTION B

- INPUTS TO RIGHT OFFSPRING ARE A SUBSET OF THE INPUTS TO THE PARENT.
- INPUTS TO LEFT OFFSPRING COME FROM THE OUTPUTS OF THE RIGHT OFFSPRING AND FROM THE INPUTS TO THE PARENT; NONE MUST NECESSARILY COME FROM THE PARENT.
- OUTPUTS FROM THE PARENT COME FROM THE OUTPUTS OF THE LEFT AND RIGHT OFFSPRING. EACH OFFSPRING CONTRIBUTES AT LEAST ONE OUTPUT OF THE PARENT.

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

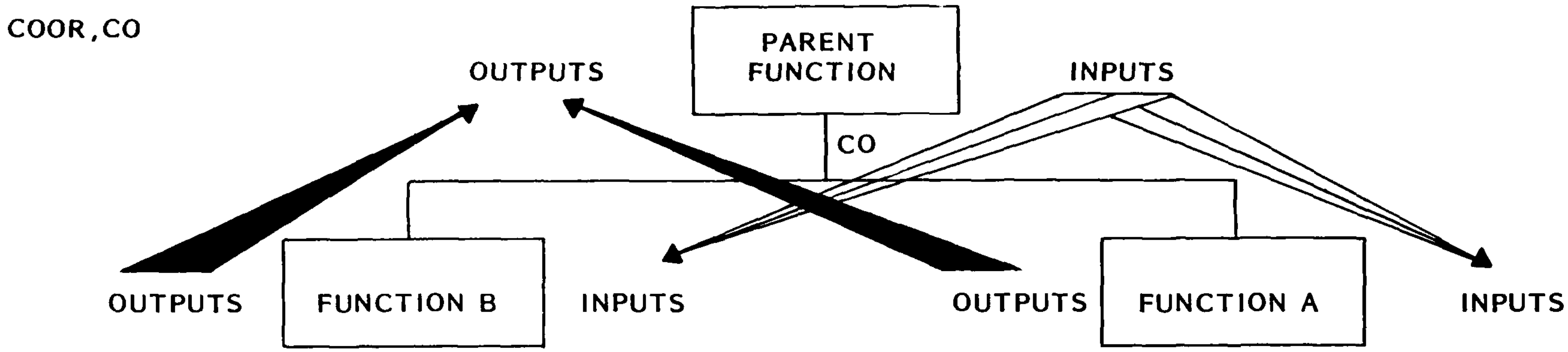
INSTRUCTOR NOTES

FOR COOR,



NON-PRIMITIVE CO CONTROL STRUCTURE

(COOR)



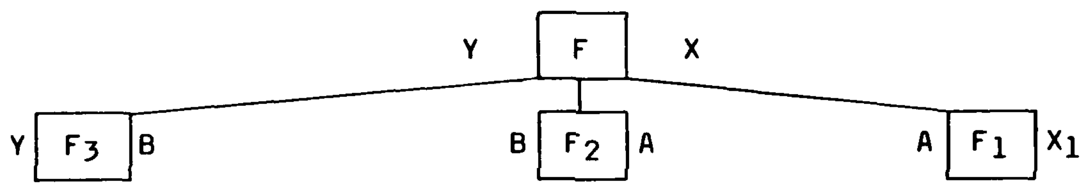
FUNCTION A OR FUNCTION B IS PERFORMED

- INPUTS TO BOTH OFFSPRINGS ARE A SUBSET OF THE INPUTS TO THE PARENT.
  - A PARTITION FUNCTION P USES SOME INPUT FROM THE PARENT TO DETERMINE WHICH OFFSPRING FUNCTION IS USED.
- OUTPUTS FROM ALL FUNCTIONS ARE IDENTICAL.

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

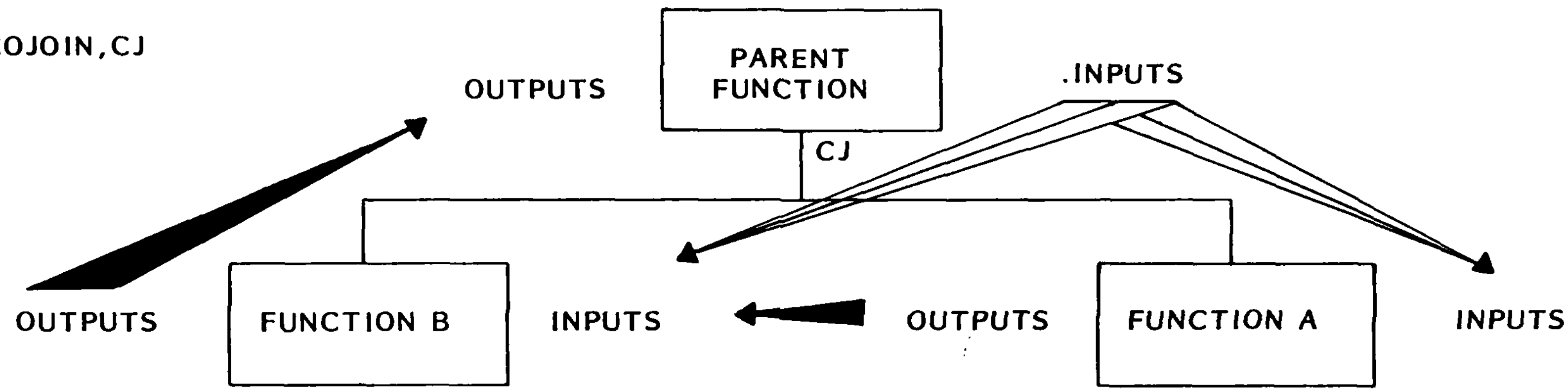
FOR COJOIN,



NON-PRIMITIVE CO CONTROL STRUCTURE

(COJOIN)

COJOIN,CJ

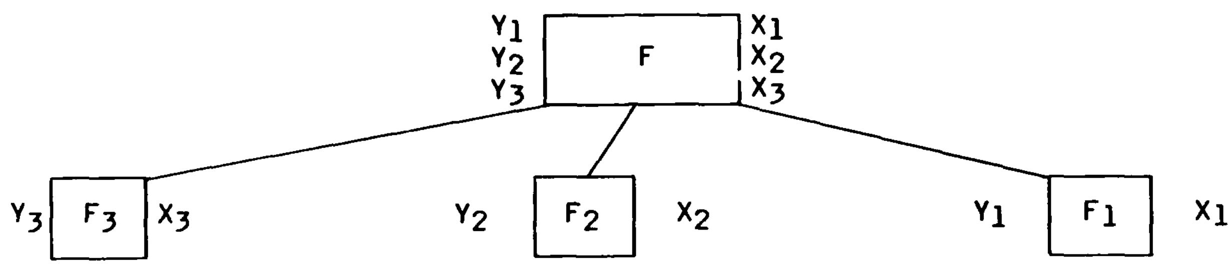


FUNCTION A IS PERFORMED FIRST, THEN FUNCTION B

- INPUTS TO RIGHT OFFSPRING ARE A SUBSET OF THE PARENT'S INPUTS.
- INPUTS TO LEFT OFFSPRING COME FROM EITHER THE PARENT'S INPUTS OR THE
  - OUTPUTS OF THE RIGHT OFFSPRING.
- OUTPUTS FROM PARENT ARE IDENTICAL TO THE OUTPUTS FROM THE LEFT OFFSPRING.

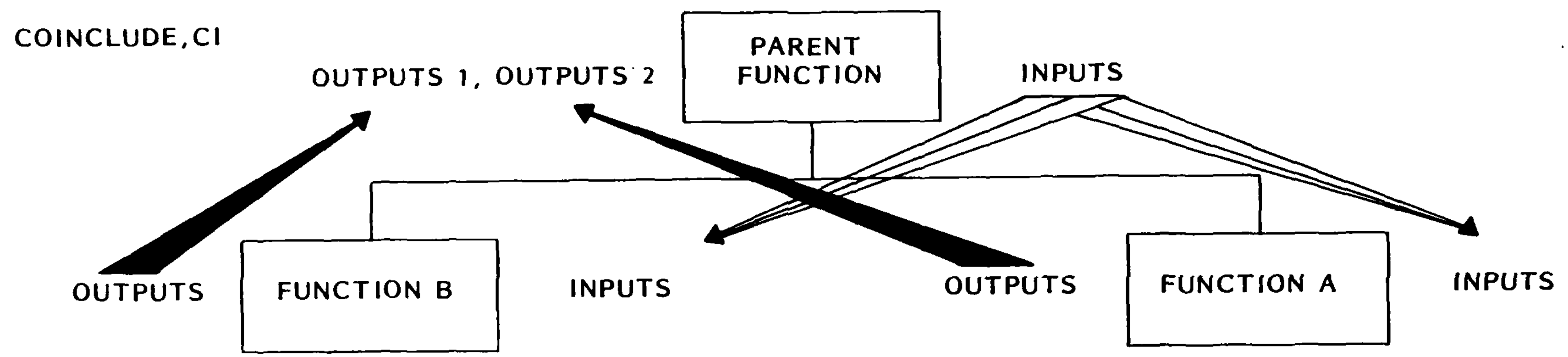
SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

FOR COINCLUDE,



NON-PRIMITIVE CO CONTROL STRUCTURE

(COINCLUDE)



FUNCTION A AND FUNCTION B ARE BOTH PERFORMED

- INPUTS TO RIGHT AND LEFT OFFSPRINGS COME FROM THE PARENT'S INPUTS.
- ALL OUTPUTS FROM PARENT COME FROM EITHER THE OUTPUTS OF THE LEFT OFFSPRING OR THE RIGHT OFFSPRING.
  - THE FIRST OUTPUTS OF THE PARENT ARE IDENTICAL TO THOSE OF THE LEFT OFFSPRING.
  - THE LAST OUTPUTS OF THE PARENT ARE IDENTICAL TO THOSE OF THE RIGHT OFFSPRING.

SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

HOW TO READ HOS CHARTS

HOS HIERARCHY

- FUNCTIONS ARE DECOMPOSED IN A TOP DOWN MANNER.

- DECOMPOSITION CONTINUES UNTIL THE "LEAF" NODES OF THE HIERARCHY ARE PRIMITIVE OPERATIONS OR SUBROUTINES THAT HAVE BEEN PREVIOUSLY PROVED CORRECT.

- FOUR TYPES OF "LEAF" NODES
  - P: PRIMITIVE OPERATION - AN OPERATION THAT CANNOT BE DECOMPOSED INTO OTHER OPERATIONS. IT IS DEFINED RIGOROUSLY WITH MATHEMATICAL AXIOMS.
  - OP: OPERATION DEFINED ELSEWHERE - A FUNCTION THAT WILL BE FURTHER DECOMPOSED IN ANOTHER CONTROL MAP, WHICH MAY BE PART OF THE CURRENT DESIGN OR MAY BE IN A LIBRARY.
  - R: RECURSIVE OPERATION - A SPECIAL NODE THAT ALLOWS LOOPING.
  - XO: EXTERNAL OPERATION - A FUNCTION THAT IS AN EXTERNAL PROGRAM WHICH IS NOT WRITTEN WITH HOS METHODOLOGY. IT MAY BE MANUFACTURER'S SOFTWARE OR PREVIOUSLY EXISTING USER PROGRAMS. NEEDLES TO SAY, THE HOS SOFTWARE CANNOT GUARANTEE ITS CORRECTNESS.

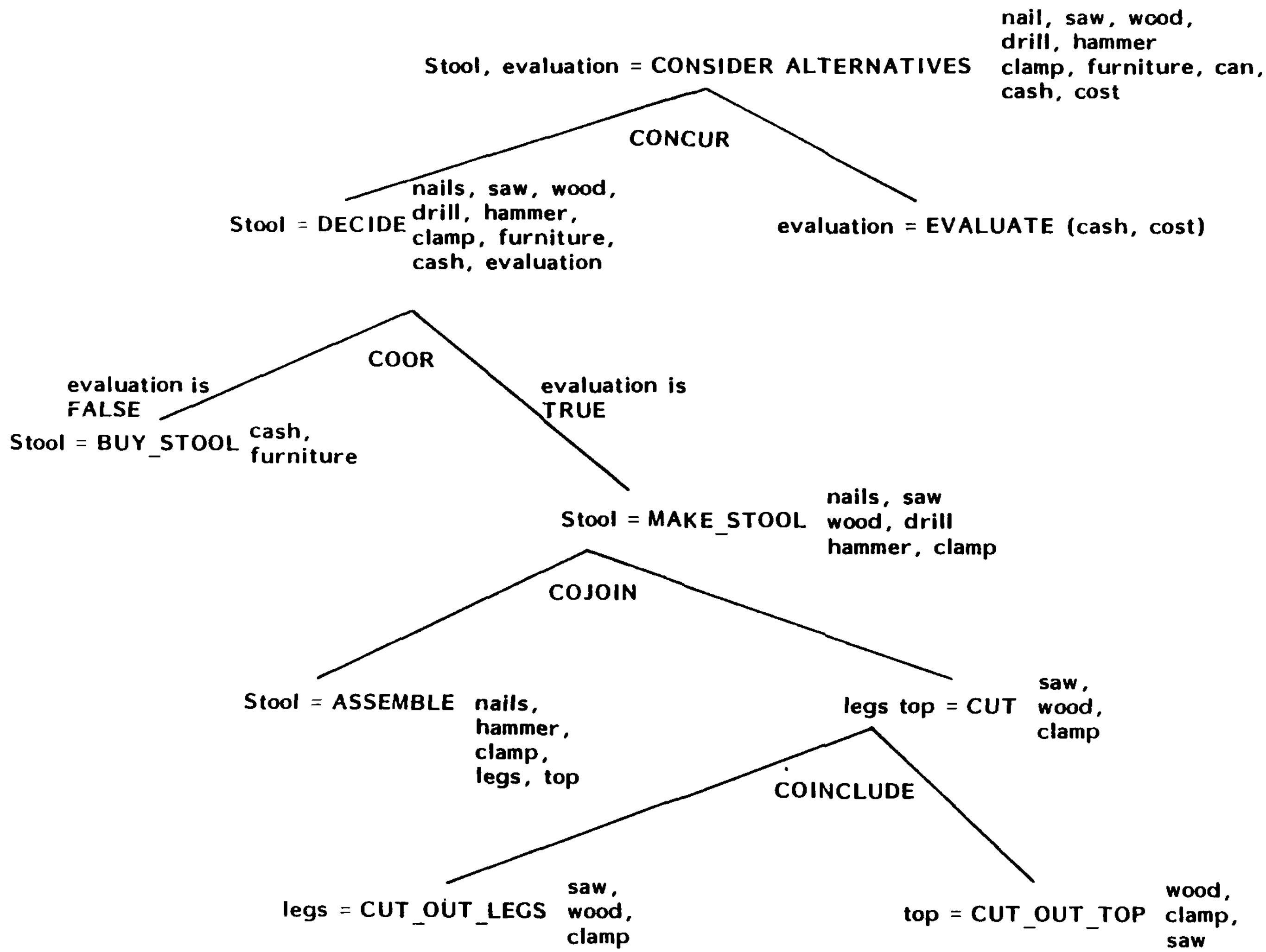
SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

EMPHASIZE THE HIERARCHICAL NATURE OF THE PROBLEM SOLUTION

HOS EXAMPLE

(ACQUIRE A STOOL)



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

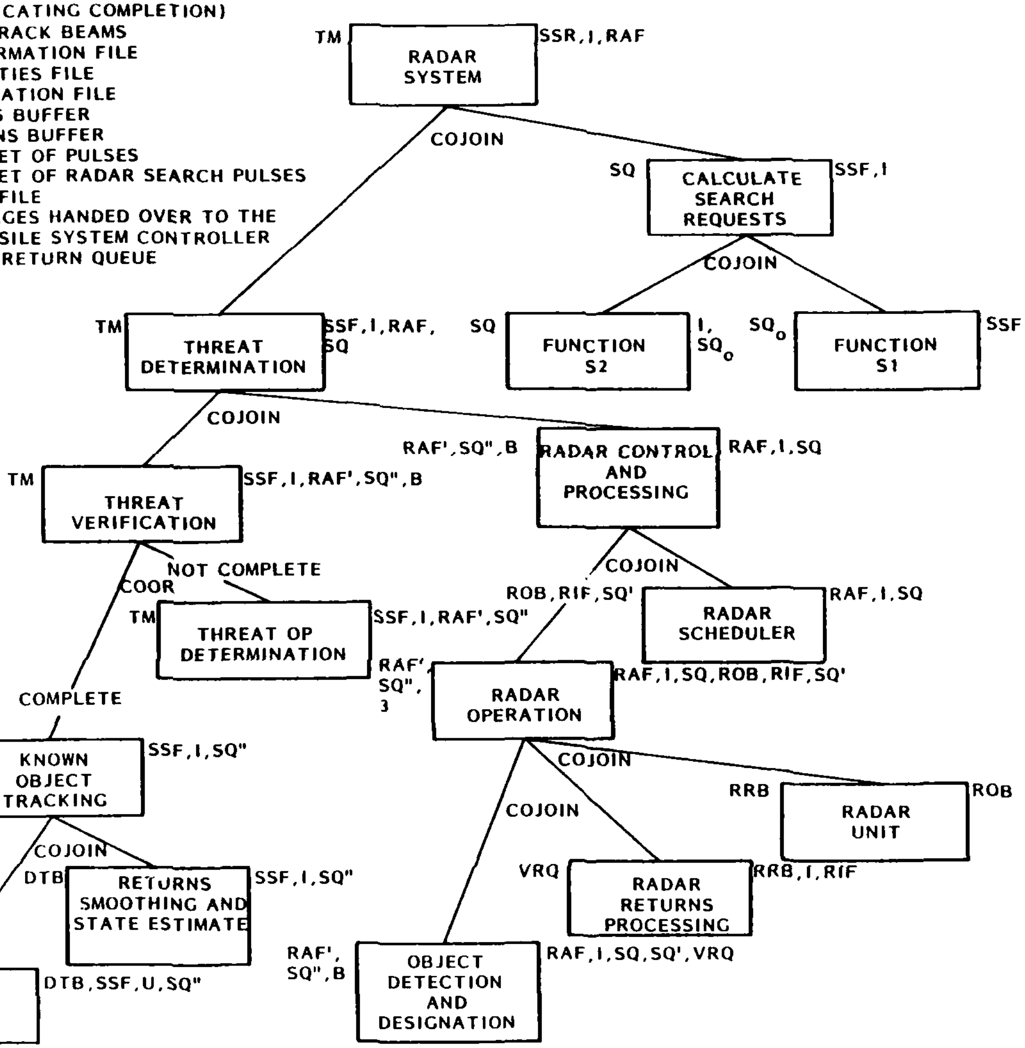
INSTRUCTOR NOTES

GO THROUGH QUICKLY. NEXT SLIDE SHOWS THREAT DETERMINATION LEAF DETAIL

RADAR SYSTEM

EXAMPLE

B:	BOOLEAN (INDICATING COMPLETION)
DTB:	DESIGNATED TRACK BEAMS
I:	GENERAL INFORMATION FILE
RAF:	RADAR ACTIVITIES FILE
RIF:	RADAR INFORMATION FILE
ROB:	RADAR ORDERS BUFFER
RRB:	RADAR RETURNS BUFFER
SQ_o:	AN ORDERED SET OF PULSES
SQ:	AN ORDERED SET OF RADAR SEARCH PULSES
SSF:	SEARCH SCAN FILE
TM:	THREAT MESSAGES HANDED OVER TO THE BALLISTIC MISSILE SYSTEM CONTROLLER
VRQ:	VERIFICATION RETURN QUEUE



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

VG 778.1

INSTRUCTOR NOTES

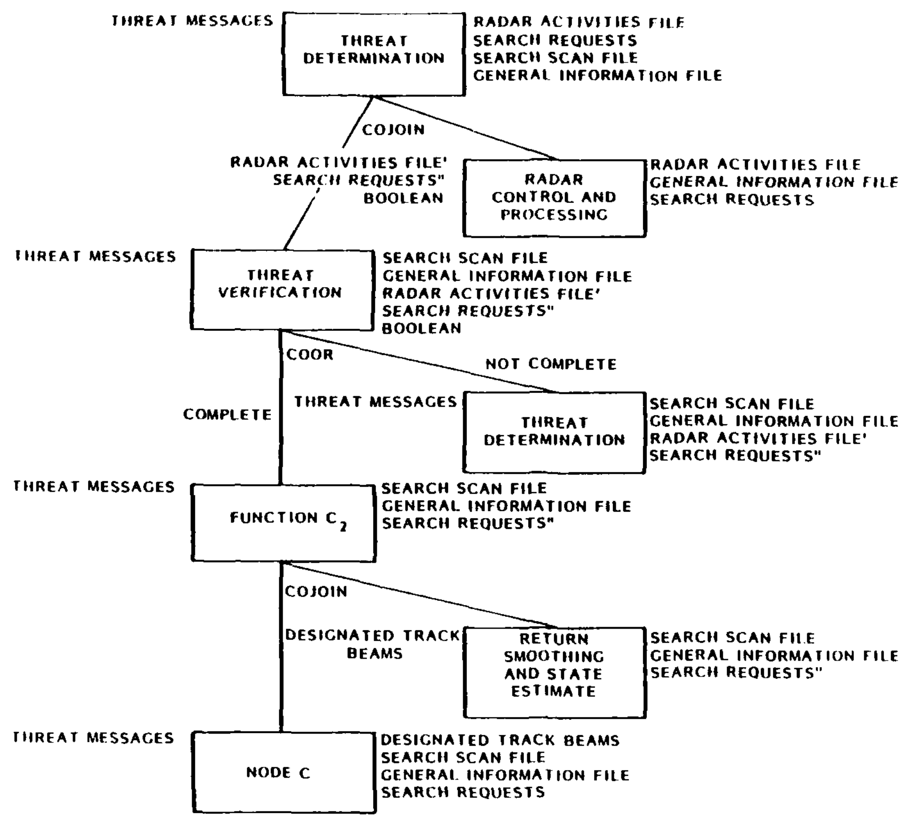
DISCUSS THE CAPABILITY OF PARTITIONING THE DESIGN EASILY

RADAR SYSTEM

EXAMPLE

(OP LEAF)

- MAY HAVE BEEN PREVIOUSLY DEVELOPED



SOURCE: MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS, 1985

INSTRUCTOR NOTES

SOFTWARE FOR APPLYING SUCH TECHNIQUES REQUIRES THE FOLLOWING COMPONENTS

1. A LANGUAGE FOR EXPRESSING FUNCTIONS AND THEIR DECOMPOSITION INTO OTHER FUNCTIONS

2. AN INTERACTIVE SCREEN FACILITY FOR CONSTRUCTING AND MANIPULATING THE CONTROL MAPS, AND ALLOWING THE USER TO CORRECT ERRORS INTERACTIVELY

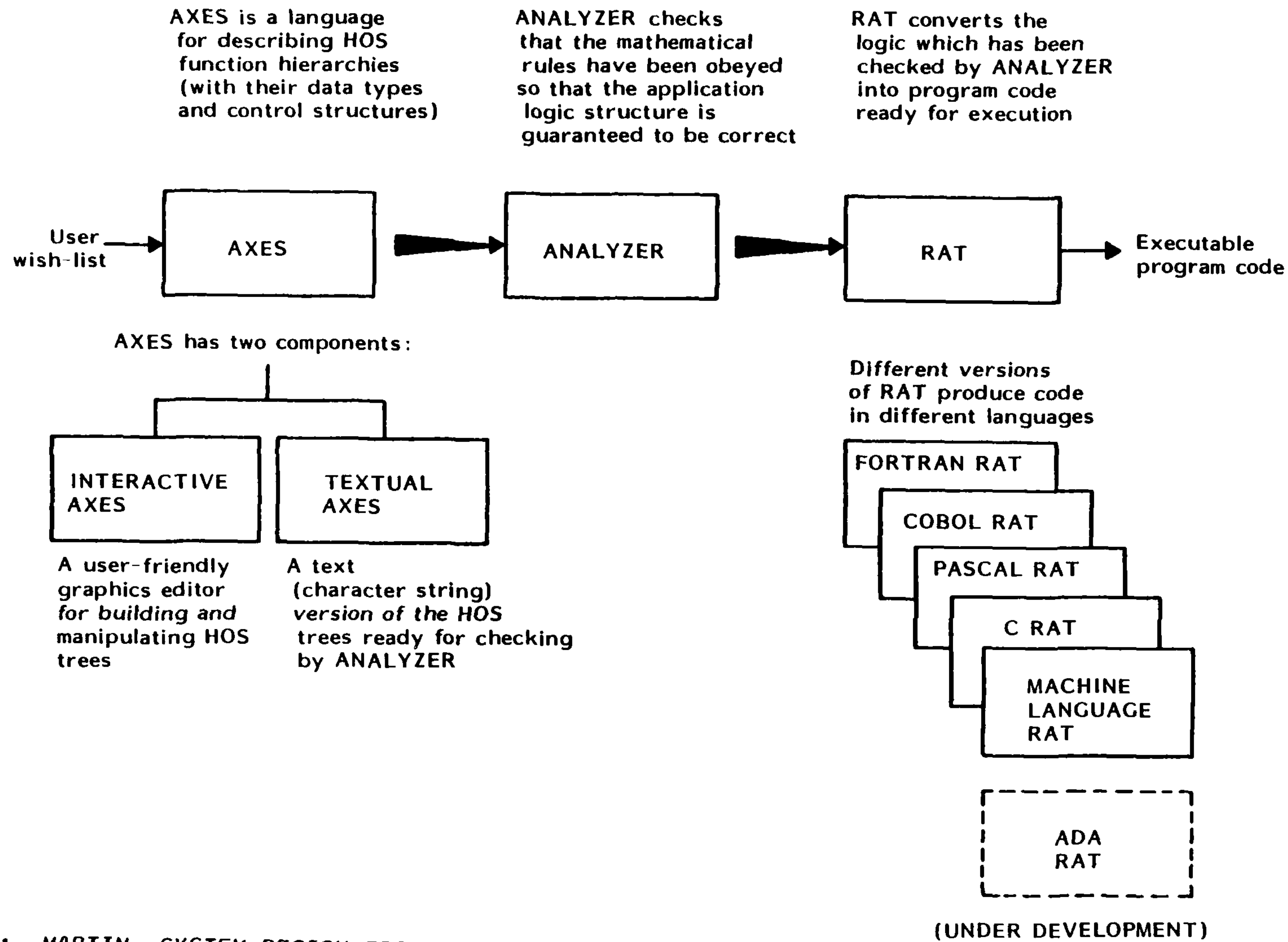
3. A LIBRARY OF DATA TYPES, PRIMITIVE FUNCTIONS, AND PREVIOUSLY DEFINED MODULES

4. AN ANALYZER ROUTINE FOR AUTOMATICALLY CHECKING THAT ALL THE RULES THAT GIVE PROVABLY CORRECT LOGIC HAVE BEEN FOLLOWED

5. A GENERATOR THAT AUTOMATICALLY GENERATES PROGRAM CODE

AUTOMATION OF HOS

(USE.IT)



SOURCE: *MARTIN, SYSTEM DESIGN FROM PROVABLY CORRECT CONSTRUCTS*, 1985

INSTRUCTOR NOTES

THEME: DESIGN METRICS GIVE US THE ABILITY TO ASSESS THE "QUALITY" OF PROPOSED DESIGNS. DESIGN METRIC CAN BE UTILIZED WITH ANY OF THE PREVIOUS METHODS.

PURPOSE: TO SHOW HOW TO MEASURE THE QUALITY OF A DESIGN.

REFERENCES:

CONSTANTINE, L., YOURDON, E. "STRUCTURED DESIGN", PRENTICE-HALL, NJ; 1978

Section 20

ARCHITECTURAL DESIGN METRICS

INSTRUCTOR NOTES

DISCUSS THE PROBLEM OF MEASURING QUALITY

ARCHITECTURAL DESIGN METRICS

QUALITY FACTORS

- QUALITY MEANS DIFFERENT THINGS TO DIFFERENT PEOPLE
  - GENERALITY
  - FLEXIBILITY
  - RELIABILITY
  - REUSABILITY

- THE ARCHITECTURAL DESIGN OF SOFTWARE DETERMINES TO A LARGE EXTENT THE "QUALITY" OF THE RESULTING IMPLEMENTATION
  - IMPORTANT TO HAVE METRICS TO ASSESS "QUALITY"

- CANDIDATE METRICS
  - COUPLING: MEASURES RELATIONSHIPS AMONG MODULES
  - COHESION: MEASURES RELATEDNESS WITHIN A MODULE
  - DESIGN HEURISTIC: MEASURES VARIOUS ASPECTS (INPUT/OUTPUT DENSITY)

INSTRUCTOR NOTES

WE WILL DISCUSS LOW COUPLING ON NEXT SLIDE

DEFINITION OF COUPLING

- COUPLING IS A MEASURE OF THE RELATIONSHIPS/INTERFACES AMONG MODULES.
- MODULE RELATIONSHIPS ARE AN INDICATION OF "QUALITY" IN A SYSTEM/SUBSYSTEM, TO THE EXTENT THAT RELATIONSHIPS ARE MINIMIZED TO INCREASE MODULE INDEPENDENCE THROUGH THE SIMPLICITY OF INTERFACES.
- MODULE RELATIONSHIPS ARE MINIMIZED BY IDENTIFYING THE FORMS OF COUPLING ON A STRUCTURE CHART AND BY DECREASING THE COUPLING WHENEVER POSSIBLE.
- COUPLING CAN BE USED AS A QUALITY METRIC ASSOCIATED WITH ANY OF THE ARCHITECTURAL DESIGN METHODS BUT USUALLY WITH STRUCTURED DESIGN.

INSTRUCTOR NOTES

EMPHASIZE DESIRABLE = LOW.

ALSO, WE WILL SHOW LEAST DESIRABLE FIRST, SO WE CAN UNDERSTAND WHY LOW COUPLING IS BETTER.

CLASSES OF COUPLING

PRIORITIZED LEVELS

<u>QUALITY FACTOR</u>	<u>CLASS</u>	<u>DEGREE OF COUPLING</u>
UNDESIRABLE		HIGH
	CONTENT	
	COMMON	
	EXTERNAL	
	CONTROL	
	STAMP	
	DATA	
DESIRABLE		LOW

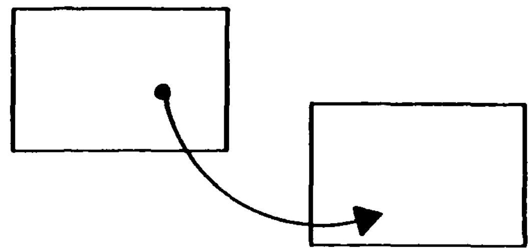
INSTRUCTOR NOTES

MODERN HIGH ORDER LANGUAGES TRY TO PROHIBIT THESE. THEY SHOULD BE AVOIDED. ASK WHY.

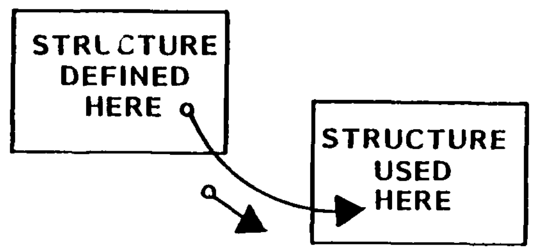
CONTENT COUPLING

- DIRECT REFERENCE INSIDE ANOTHER MODULE ...

  - INVOCATION (BRANCH) THE STATEMENT BRANCHED TO IS <u>NOT</u> AN EXTERNAL ENTRY POINT.

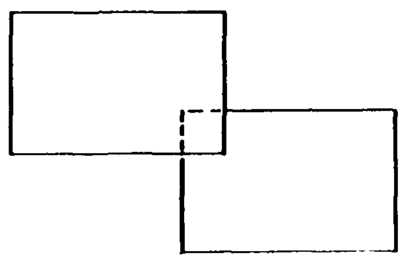


  - DATA-STRUCTURED CONNECTION



(NON-EXTERNALLY DECLARED DATA)

  - SHARING THE SAME CONTENTS (DATA)



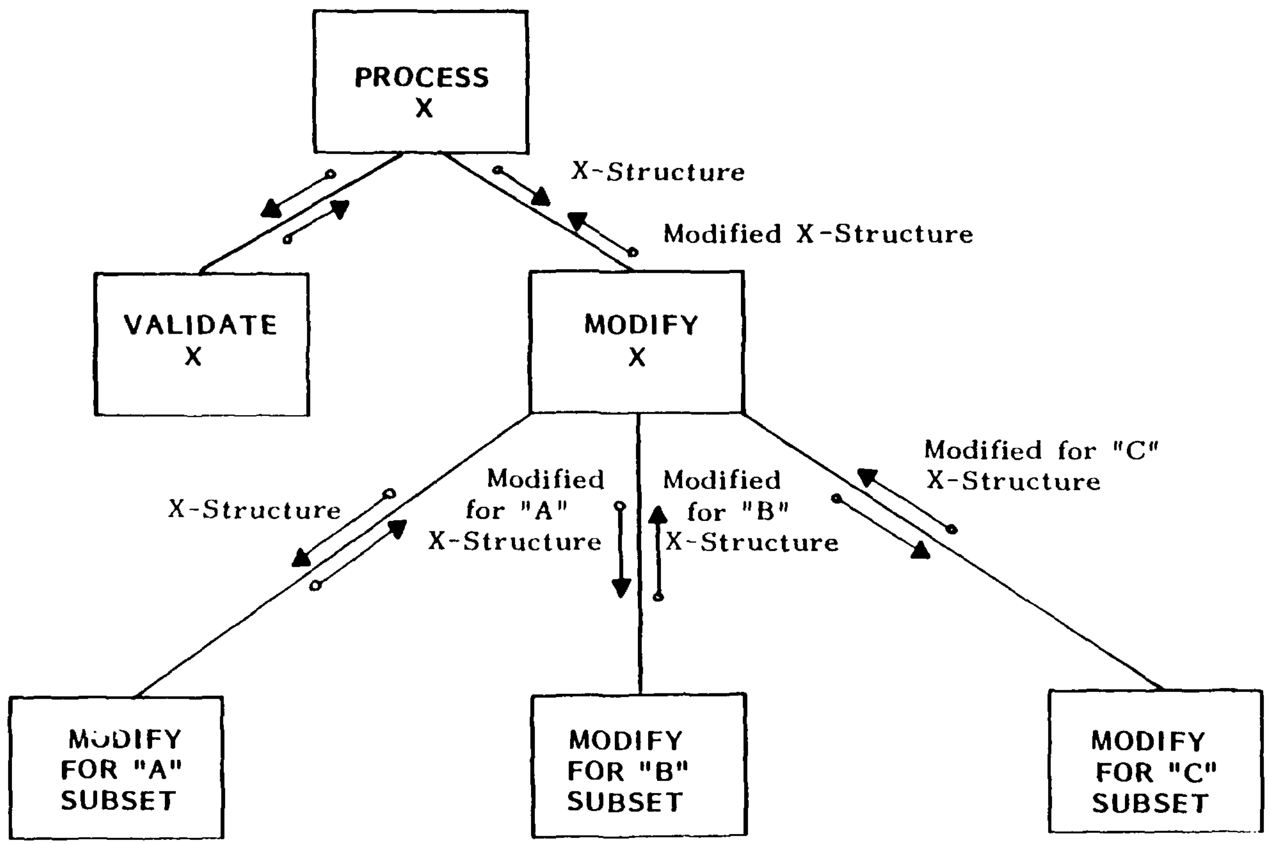
(SUBROUTINE SHARING NON-EXTERNALLY DECLARED DATA)

INSTRUCTOR NOTES

THIS IS GLOBALLY SHARED DATA. THIS IS TIGHT COUPLING.

COMMON COUPLING

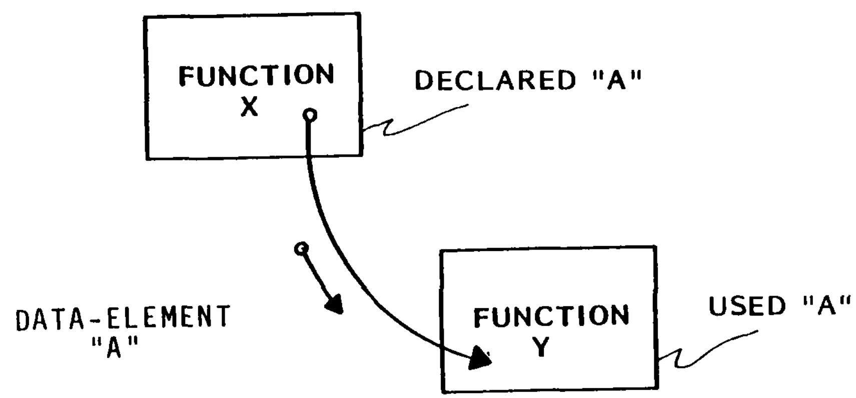
- SHARED GLOBAL DATA STRUCTURE ...



INSTRUCTOR NOTES

EXTERNAL COUPLING

- EXTERNALLY DECLARED SYMBOL:



INSTRUCTOR NOTES

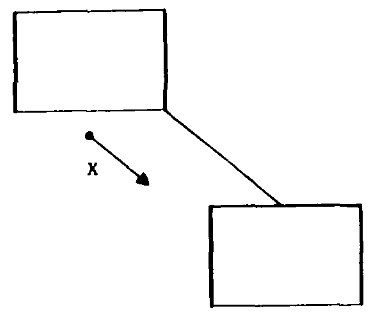
STATE THAT DOWNWARD (WHILE IT SHOULD BE MINIMIZED) IS NOT AS BAD AS UPWARD (WHICH SHOULD BE ELIMINATED).

DISCUSS THE ADVANTAGES AND DISADVANTAGES OF LOCAL VS. GLOBAL SCOPE.

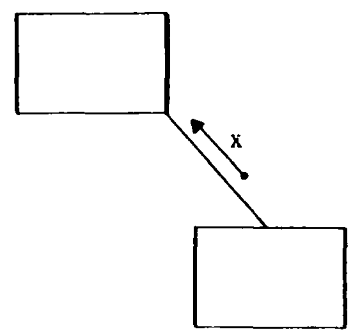
CONTROL COUPLING

- TWO KINDS OF CONTROL COUPLING
  - RADICAL : BRIDGES SEVERAL LEVELS
  - LOCAL : AT A SINGLE LEVEL

- TWO SITUATIONS IN WHICH CONTROL COUPLING OCCURS
  - DOWNWARDS FLOW (REDUCE)



  - UPWARDS FLOW (ELIMINATE)

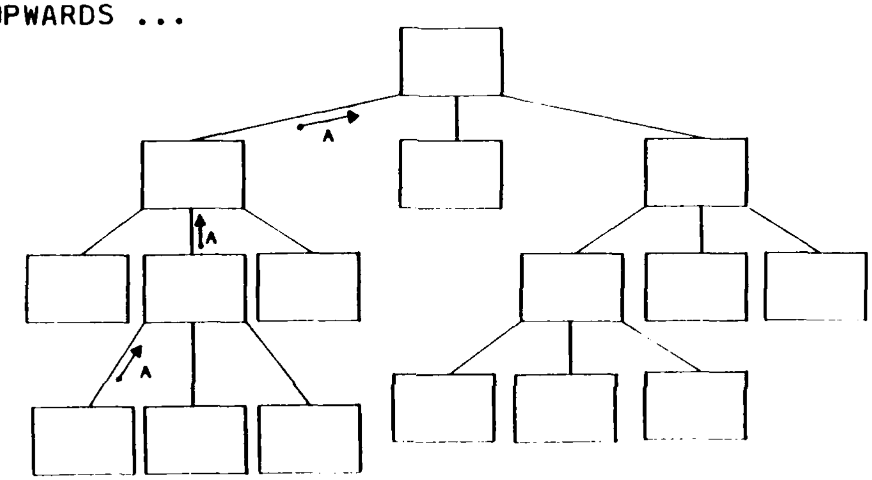


INSTRUCTOR NOTES

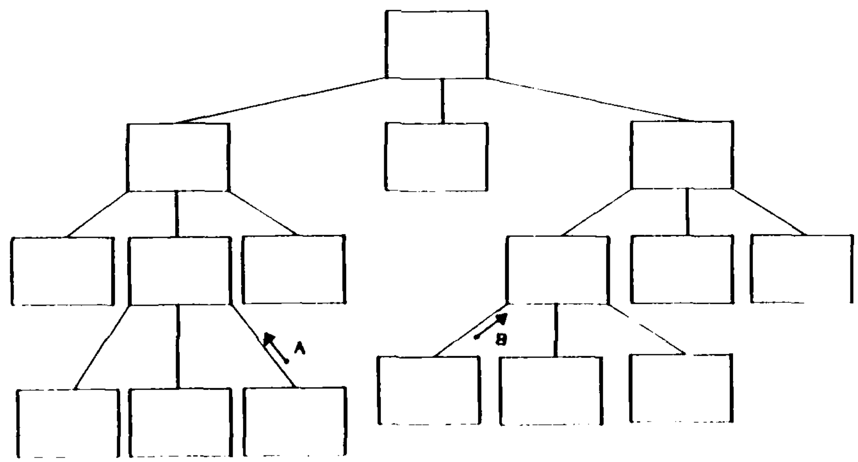
RADICAL IS *UPWARDS VERY UNDESIRABLE*. *THIS USUALLY HAPPENS AS A RESULT OF FAILING* TO ALLOW FOR SOMETHING INITIALLY.

CONTROL COUPLING

- RADICAL UPWARDS ...



- LOCAL UPWARDS ...

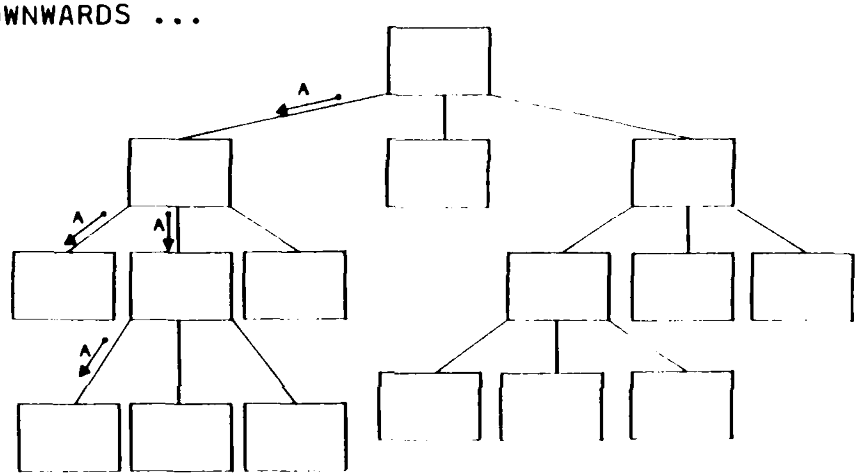


INSTRUCTOR NOTES

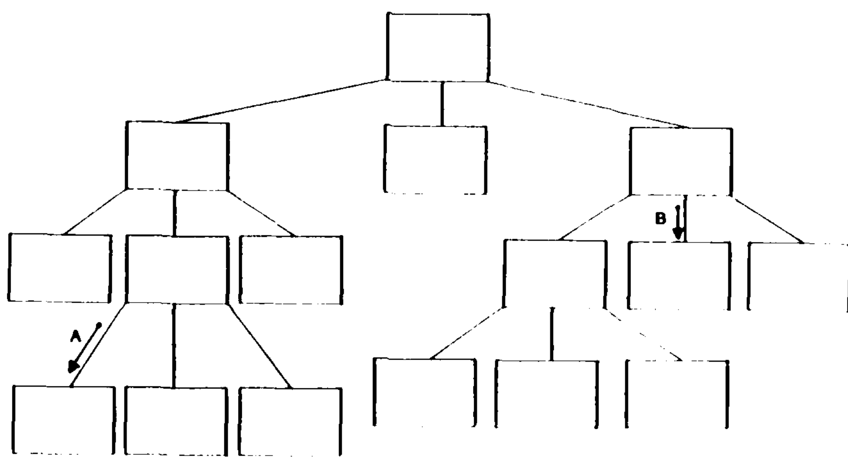
RADICAL DOWNWARDS IS UNDESIRABLE.

CONTROL COUPLING

- RADICAL DOWNWARDS ...



- LOCAL DOWNWARDS ...

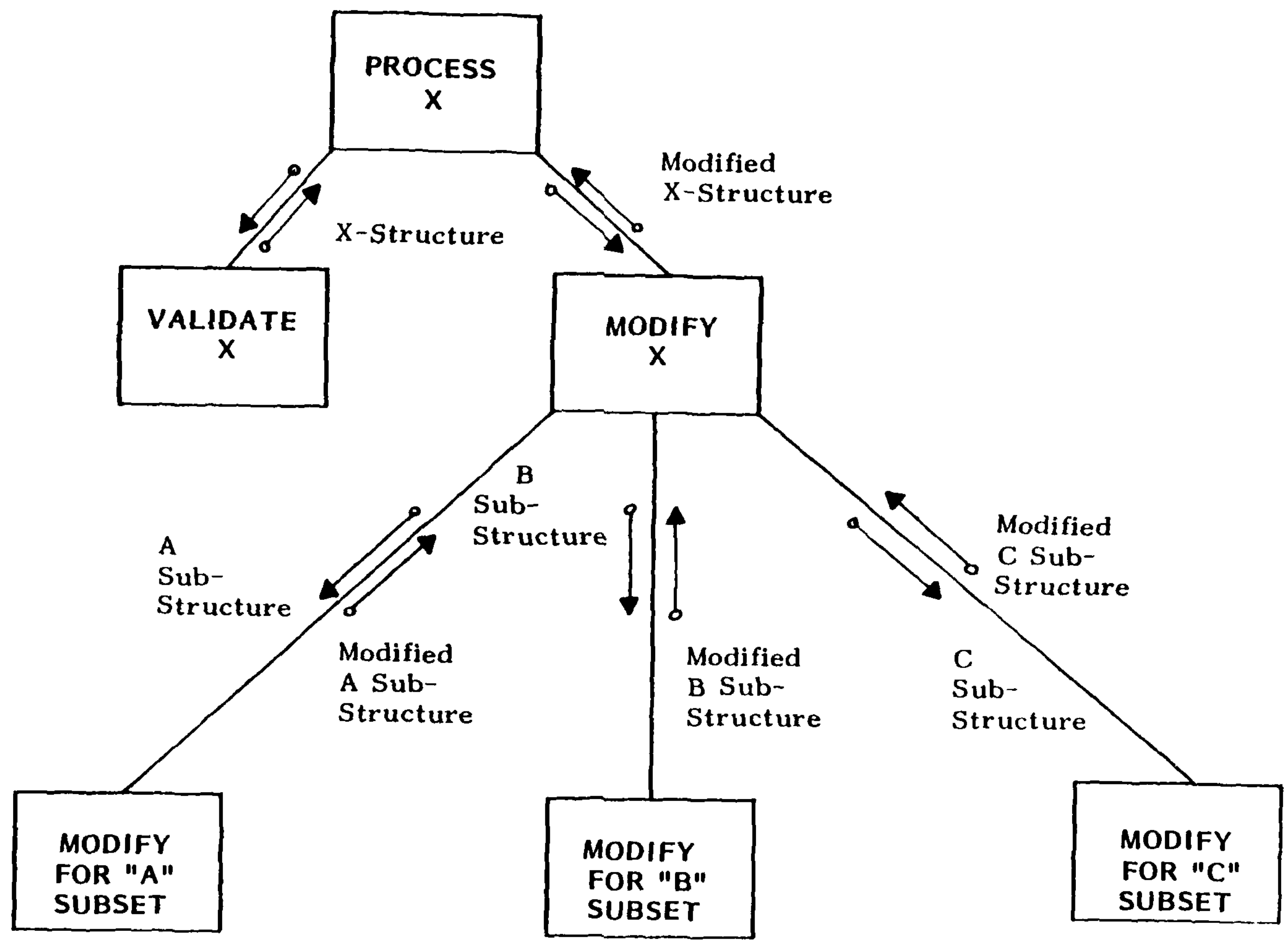


INSTRUCTOR NOTES

HERE IS COMMUNICATION THROUGH A GLOBAL DATA STRUCTURE WHERE ONLY THAT PORTION OF THE DATA NEEDED IS PASSED. ONLY A SUBSTRUCTURE IS PASSED DOWN IN THIS EXAMPLE. REFER BACK TO COMMON COUPLING AND COMPARE THE TWO KINDS OF COUPLING.

STAMP COUPLING

- GLOBAL DATA STRUCTURE ...

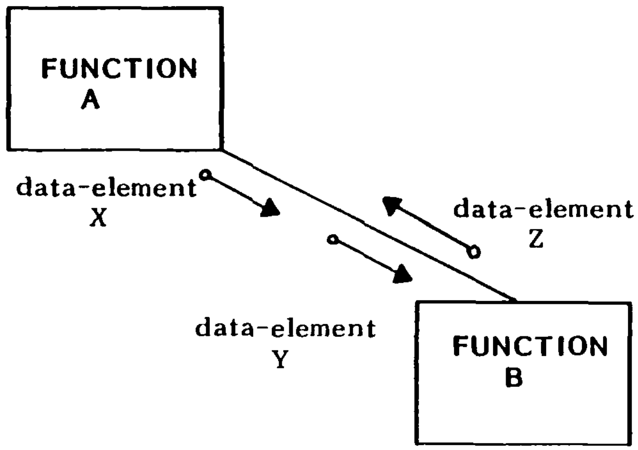


INSTRUCTOR NOTES

INTERFACES BETWEEN MODULES ARE THROUGH EXPLICIT PARAMETERS.

DATA COUPLING

- EXPLICIT ARGUMENT LISTS ...

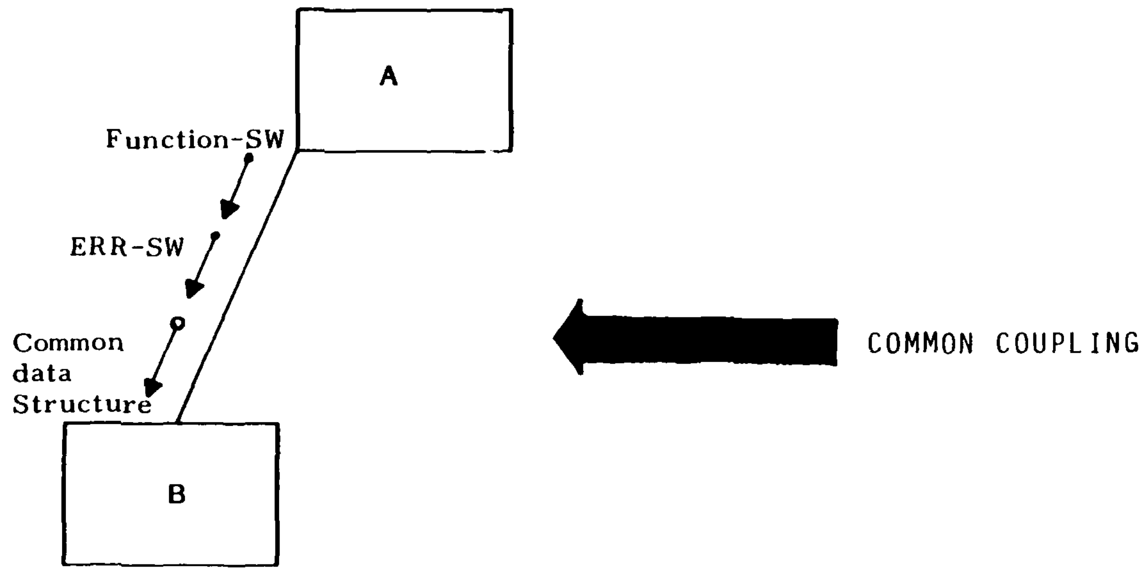


INSTRUCTOR NOTES

ONE INTERFACE MAY CONTAIN MORE THAN ONE KIND OF COUPLING.

MULTIPLE LEVELS OF COUPLING

- MULTIPLE LEVELS? PICK THE HIGHEST CLASS OF COUPLING.



INSTRUCTOR NOTES

COUPLING

SUMMARY OF USAGE

- IDENTIFY CLASSES OF COUPLING IN A DESIGN THEN CATEGORIZE THEM AS ...

UNDESIRABLE (IMPLICIT INTERFACES)		ACCEPTABLE OR DESIRABLE (EXPLICIT INTERFACES)	
CONTENT COUPLING	NON-EXTERNALLY DECLARED DATA	CONTROL COUPLING	ELEMENTS OF CONTROL PASSED THROUGH AN ARGUMENT LIST
COMMON COUPLING	EXTERNALLY DECLARED DATA-STRUCTURE "COMMON ENVIRONMENT"	STAMP COUPLING	LIMITED DATA-STRUCTURE PASSED THROUGH AN ARGUMENT LIST
EXTERNAL COUPLING	EXTERNALLY DECLARED DATA-ELEMENT	DATA COUPLING	DISCRETE DATA ELEMENTS PASSED THROUGH AN ARGUMENT LIST

- RESTRUCTURE DESIGN TO ELIMINATE UNDESIRABLE CLASSES OF COUPLING.

INSTRUCTOR NOTES

REVIEW AND CONTRAST TO COUPLING.

DEFINITION OF COHESION

- COHESION IS A MEASURE OF THE STRENGTH OR FUNCTIONAL RELATEDNESS OF THE ELEMENTS WITHIN A MODULE.

- MODULE STRENGTH IS AN INDICATION OF "QUALITY" IN A MODULE TO THE EXTENT THAT COMPLEXITY IS *REDUCED AND SIMPLICITY IS INCREASED THROUGH* FUNCTIONAL RELATEDNESS.

- MODULE STRENGTH IS ACHIEVED BY IDENTIFYING THE LEVEL OF COHESION ON A STRUCTURE CHART AND BY MAKING THE ELEMENTS WITHIN A MODULE CLOSELY RELATED AS POSSIBLE.

INSTRUCTOR NOTES

STRESS DESIRABLE = HIGH.

AGAIN, WE'LL GO FROM BAD TO GOOD ...

CLASSES OF COHESION

PRIORITIZED LEVELS

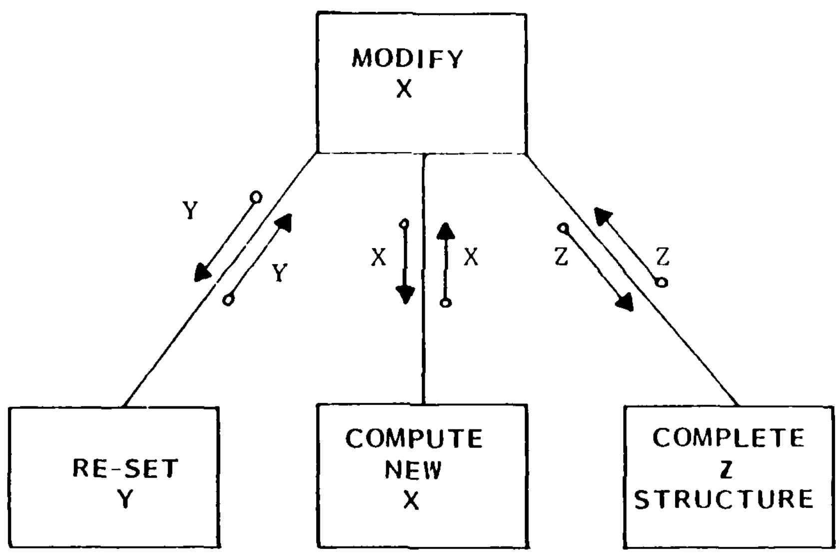
QUALITY FACTOR	CLASS	DEGREE OF COHESION
UNDESIRABLE		LOW
	COINCIDENTAL	
	LOGICAL	
	CLASSICAL	
	PROCEDURAL	
	COMMUNICATIONAL	
	INFORMATIONAL	
	FUNCTIONAL	
DESIRABLE		HIGH

INSTRUCTOR NOTES

THESE ARE GROUPED TOGETHER FOR NO OBVIOUS REASONS AT ALL. THE 3 SUBORDINATE MODULES ARE NOT RELATED TO MODIFY X.

COINCIDENTAL

- NO MEANINGFUL RELATIONSHIPS ...

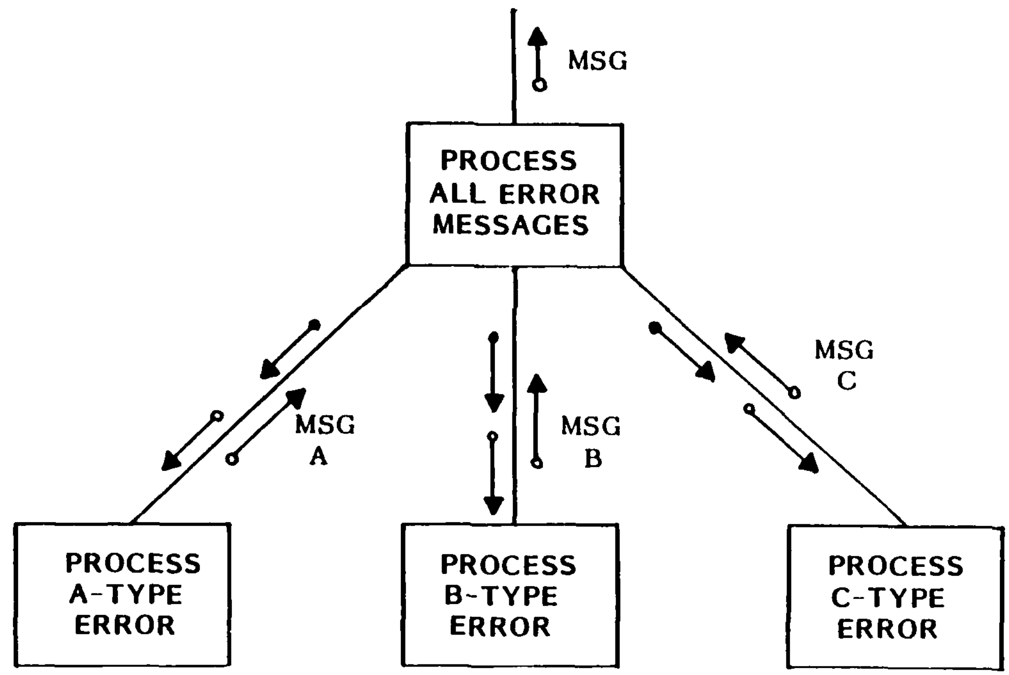


INSTRUCTOR NOTES

THE MODULES ARE RELATED BECAUSE THEY DO SIMILAR THINGS.

LOGICAL

- CLASS OF RELATED MODULES ...

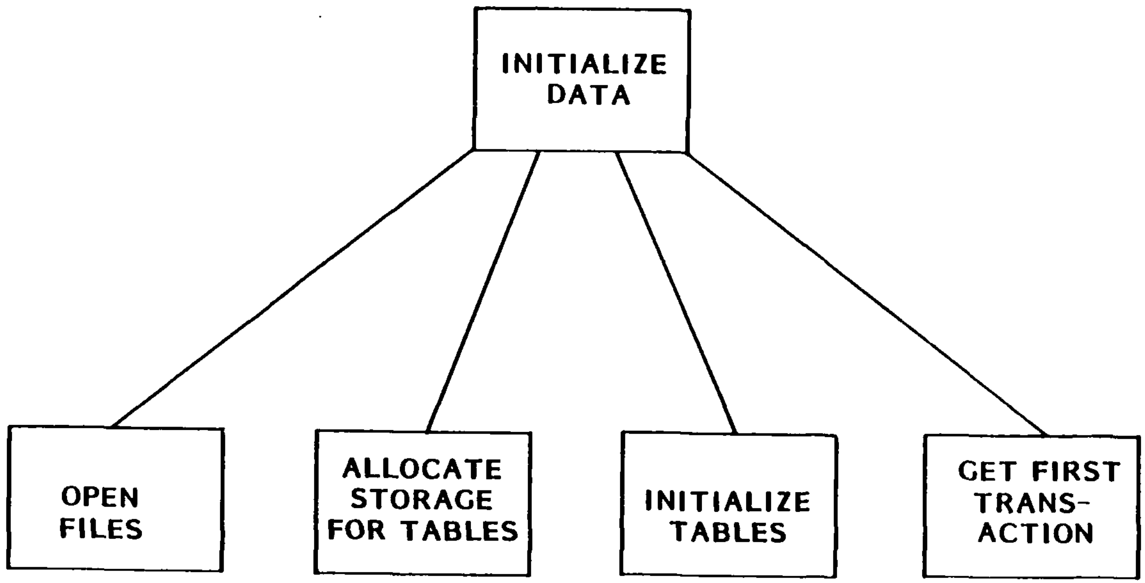


INSTRUCTOR NOTES

"TEMPORAL COHESION" ...

CLASSICAL

- TIME RELATED MODULES ...

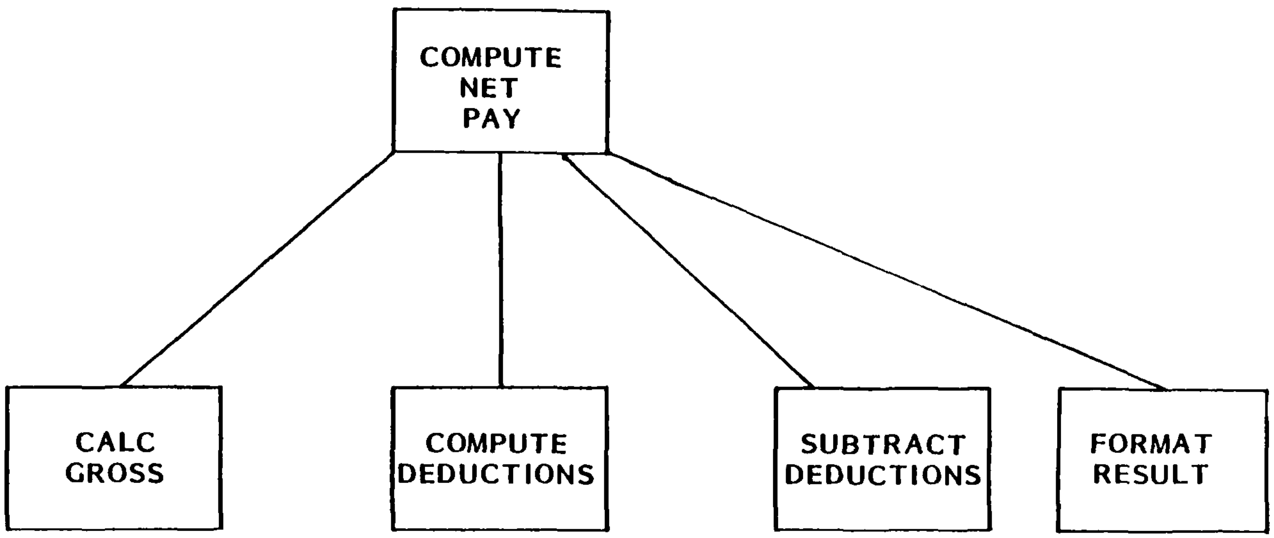


INSTRUCTOR NOTES

"FORMAT RESULT" HAS NOTHING TO DO WITH "COMPUTE NET PAY," THUS PROCEDURAL COHESION.

PROCEDURAL

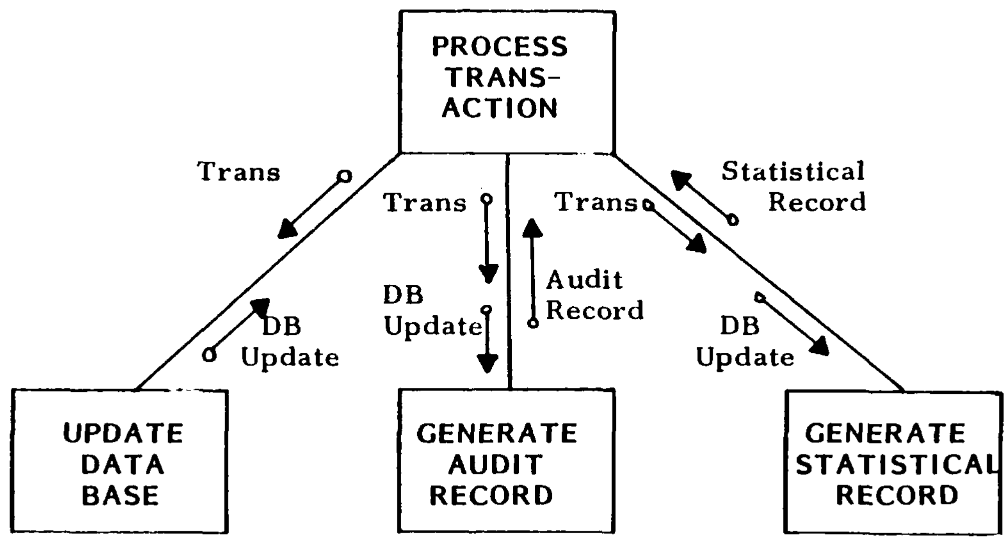
- PROBLEM RELATED MODULES ...



INSTRUCTOR NOTES

COMMUNICATIONAL

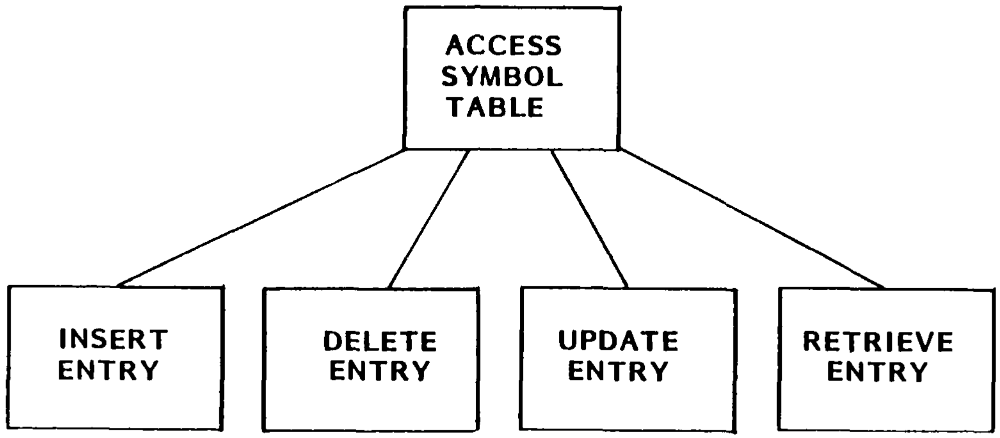
- PROCEDURAL MODULES THAT SHARE THE SAME DATA ...



INSTRUCTOR NOTES

PARNAS DATA HIDING PRODUCES INFORMATIONAL COHESION. BOTTOM MODULES ARE FUNCTIONAL, TOP MODULE IS INFORMATIONAL.

- PACKAGING FUNCTIONAL MODULES ...

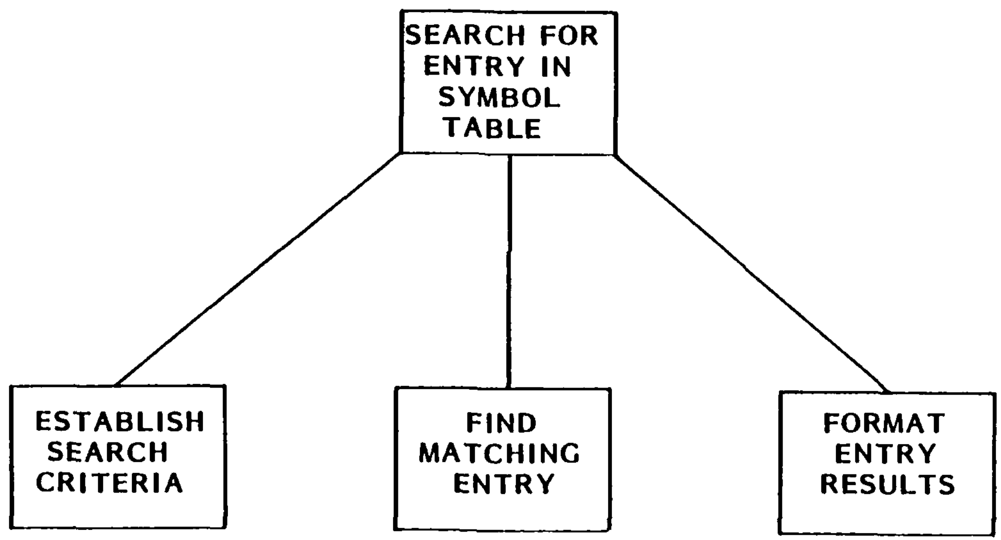


INSTRUCTOR NOTES

THIS IS THE HIGHEST LEVEL. EACH MODULE PERFORMS A SINGLE FUNCTION. ANY SUBORDINATE FUNCTION IS A <u>PART</u> OF THAT FUNCTION.

FUNCTIONAL

- SINGLE FUNCTION PER MODULE ...



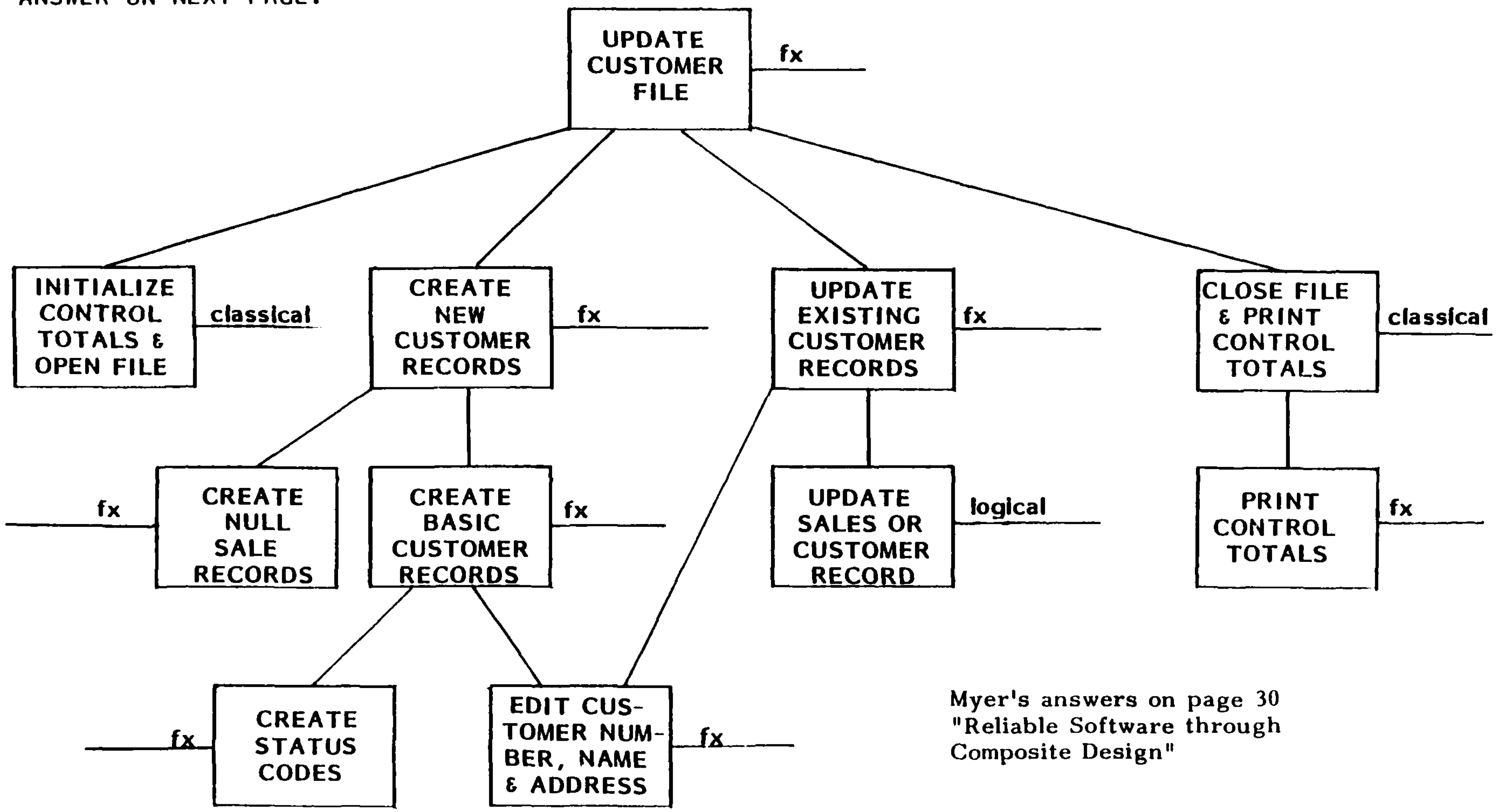
INSTRUCTOR NOTES

IDENTIFYING CLASS OF COHESION

- IDENTIFY BY THE NATURE OF THE LABEL IN THE BOX.

BOX LABEL	COHESION
VERBS CONNECTED BY "OR"	LOGICAL
VERBS CONNECTED BY "AND"	CLASSICAL, PROCEDURAL, COMMUNICATIONAL
WORDS RELATED TO TIME (FIRST, NEXT, THEN, AFTER, ETC.)	PROCEDURAL
PLURAL OBJECT (EDIT ALL DATA, PRODUCE REPORTS)	LOGICAL
INITIALIZE, TERMINATION, CLEAN-UP	CLASSICAL

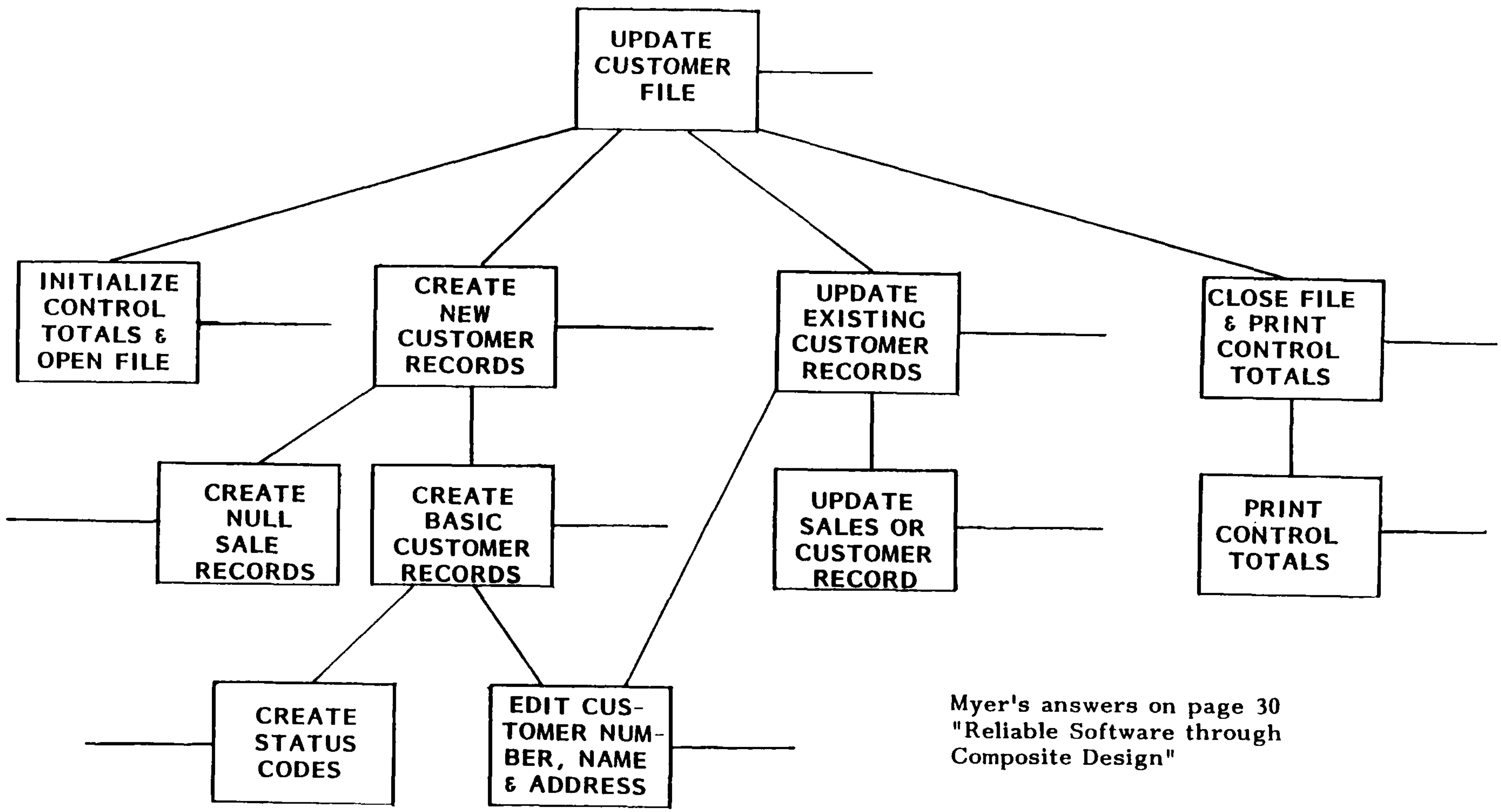
TRY TO ENCOURAGE DISCUSSION ON ALTERNATE CATEGORY FOR TOP BOX. BAIT STUDENTS, THEN POSE THE QUESTIONS "HOW MUCH TIME SHOULD BE SPENT TRYING TO DETERMINE LEVEL OF COHESION?" ANSWER ON NEXT PAGE.



Myer's answers on page 30 "Reliable Software through Composite Design"

IDENTIFY CLASS OF COHESION

EXERCISE



Myer's answers on page 30
"Reliable Software through
Composite Design"

INSTRUCTOR NOTES

THERE SHOULDN'T EXIST MORE THAN ONE TYPE OF COHESION IN A DESIGN.

COHESION
SUMMARY OF USAGE

- IDENTIFY CLASSES OF COHESION IN A DESIGN THEN CATEGORIZE THEM AS ...

UNDESIRABLE (WEAK)		ACCEPTABLE OR DESIRABLE (STRONG)	
COINCIDENTAL COHESION	NO MEANINGFUL RELATIONSHIP	PROCEDURAL COHESION	RELATED VIA THE PROBLEM
LOGICAL COHESION	CLASS OF DATA PROCESSED	COMMUNICATIONAL COHESION	COMMUNICATION THROUGH DATA
CLASSICAL COHESION	RELATED IN TIME	INFORMATIONAL COHESION	SAME DATA BEING PROCESSED BUT FUNCTIONS ARE DISTINCT
		FUNCTIONAL COHESION	SINGLE DISCRETE FUNCTION

- RESTRUCTURE DESIGN TO ELIMINATE UNDESIRABLE CLASSES OF COHESION

- FOCUS ON COUPLING BEFORE COHESION, COUPLING ...
  - IS EASIER TO IDENTIFY
  - IS EASIER TO IMPROVE
  - IS INVERSELY PROPORTIONATE TO COHESION

INSTRUCTOR NOTES

MORE CONCEPTS RELATED TO DESIGN QUALITY.

HEURISTIC: INVOLVING OR SERVING AS AN AID TO LEARNING, DISCOVERY OR PROBLEM SOLVING BY EXPERIMENTAL (ESPECIALLY TRIAL AND ERROR) METHODS.

DESIGN HEURISTICS CONSIDERED

- FAN-IN
- FAN-OUT
- SCOPE OF EFFECT/SCOPE OF CONTROL
- CONTROL STRUCTURE

INSTRUCTOR NOTES

REVIEW COUPLING AND COHESION.

FAN-IN

- FAN-IN IS THE NUMBER OF UNIQUE SUPERORDINATE MODULES THAT CALL ANOTHER MODULE.
- FAN-IN SHOULD BE MAXIMIZED TO INCREASE THE SHARING OF COMMON FUNCTIONS AND REDUCE REDUNDANT CODE.
- HIGH FAN-IN IS ACHIEVED BY SEARCHING FOR IDENTICAL OR SIMILAR FUNCTIONS AND MAKING IT A SHARED MODULE WITH HIGH FAN-IN.
- HIGH FAN-IN MODULES ...
  - HAVE HIGH COHESION
  - HAVE LOW COUPLING
  - ARE LOW IN THE CONTROL STRUCTURE

INSTRUCTOR NOTES

WHY ARE 4 SUBORDINATES MOST COMMON?

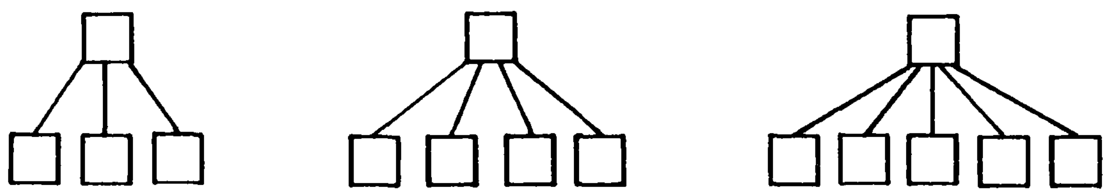
FAN-OUT

- FAN-OUT IS THE NUMBER OF UNIQUE SUBORDINATE MODULES THAT ARE CALLED BY ANOTHER MODULE.

- FAN-OUT SHOULD BE REASONABLE, USUALLY 3-6 MODULES AT ANY LEVEL.

- REASONABLE FAN-OUT IS ACHIEVED BY INTRODUCING LEVELS OF CONTROL FOR HIGH *FAN-OUT AND LEVELS OF SUBORDINATION FOR LOW FAN-OUT.*

- AROUND 4 SUBORDINATES IS MOST COMMON:



INSTRUCTOR NOTES

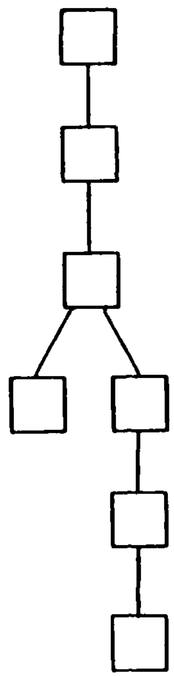
IN THE TOO LOW FAN-OUT CASE, THE MIDDLE MODULES MAY BE UNNECESSARY. IT IS BETTER TO DELEGATE FUNCTIONS AS NEEDED. A RARE CASE WHEN IT'S OK: INPUT FILTERING.

IN THE TOO HIGH FAN-OUT CASE THE TOP MODULE IS CONTROLLING TOO MUCH, AN INTERMEDIATE LEVEL OF CONTROL IS NEEDED, BUT THERE IS A RARE CASE WHEN IT'S OK, E.G., A TRANSACTION CENTER.

FAN-OUT

FAN-OUT (TOO LOW)

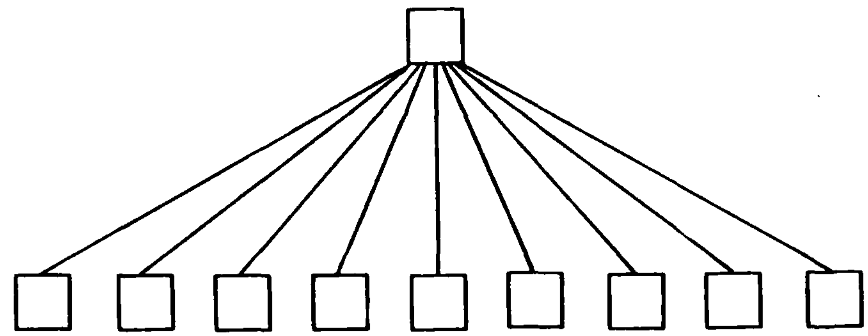
- TOO FEW SUBORDINATES CAUSE TOWER STRUCTURE ...



- AIM FOR MODERATE FAN-OUT

FAN-OUT (TOO HIGH)

- TOO MANY SUBORDINATES CAUSES PANCAKE STRUCTURE ...

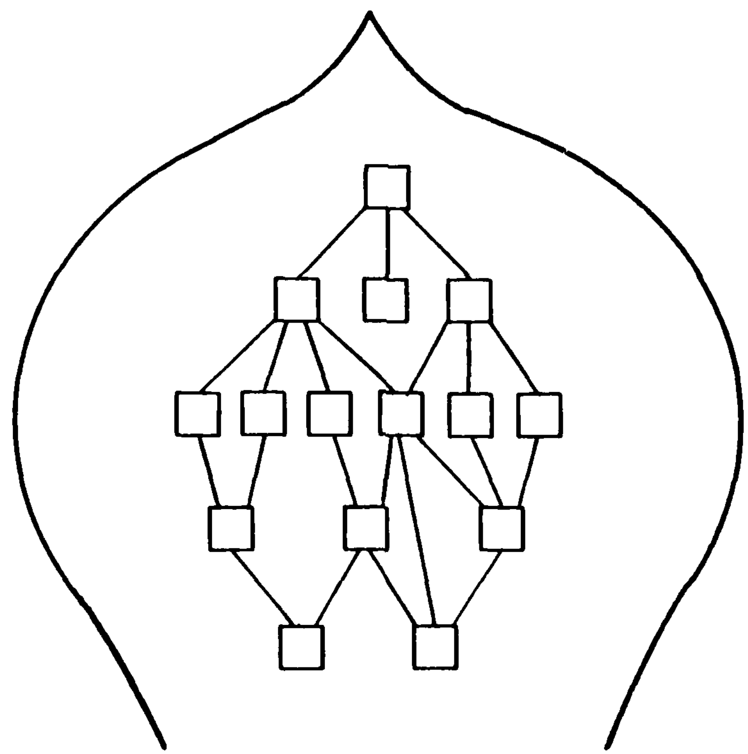


INSTRUCTOR NOTES

FAN-IN AND FAN-OUT

PERSPECTIVE

- GOOD DESIGNS HAVE A MOSQUE SHAPE ...



- HIGH FAN-IN, MODERATE FAN-OUT

INSTRUCTOR NOTES

SCOPE OF EFFECT

- SCOPE OF EFFECT (OF A DECISION) IS THE COLLECTION OF ALL MODULES THAT CONTAIN PROCESSING WHICH IS DEPENDENT ON THE RESULTS OF A DECISION.

- SCOPE OF EFFECT SHOULD BE IDENTIFIED SO THAT ONE CAN EVALUATE WHETHER OR NOT THE SCOPE OF EFFECT IS WITHIN THE SCOPE OF CONTROL.

- SCOPE OF EFFECT CAN BE IDENTIFIED ON A STRUCTURE CHART BY LOCATING ALL MODULES THAT NEED TO KNOW THE RESULTS OF THE DECISION.

INSTRUCTOR NOTES

SCOPE OF CONTROL

- SCOPE OF CONTROL OF A MODULE IS THE MODULE ITSELF AND ALL OF IT'S SUBORDINATE MODULES AT ANY LEVEL IN THE HIERARCHY.

- SCOPE OF CONTROL (OF A DECISION) FORMS A BOUNDARY WITHIN WHICH ALL SCOPES OF EFFECT MODULES SHOULD RESIDE.

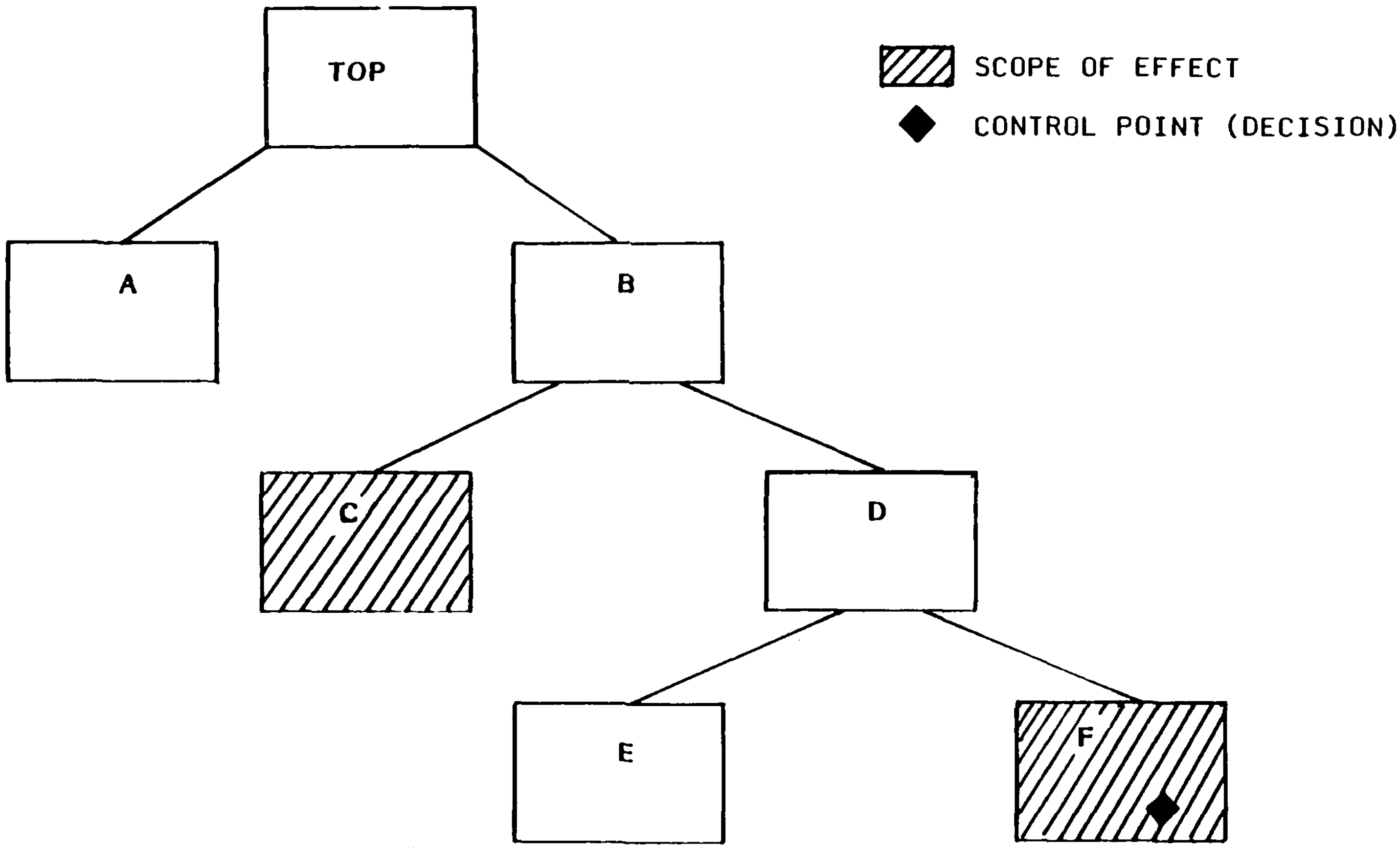
- SCOPE OF CONTROL CAN BE IDENTIFIED ON A STRUCTURE CHART BY LOCATING ALL MODULES SUBORDINATE TO AND INCLUDING THE MODULE MAKING THE DECISION.

INSTRUCTOR NOTES

CITE MODULE "C" BEING OUTSIDE REASONABLE SCOPE OF EFFECT.

SCOPE OF EFFECT/SCOPE OF CONTROL

EXAMPLE



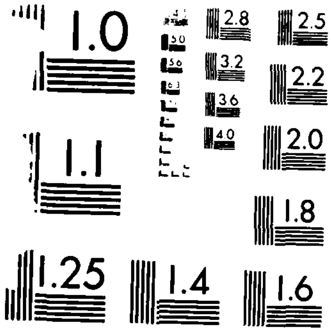
INSTRUCTOR NOTES

THE RESTRUCTURE HAS ONLY ONE LEVEL BETWEEN DECISION AND ITS EFFECT.

AD-A165 301 ADA (TRADEMARK) TRAINING CURRICULUM SOFTWARE ENGINEERING METHODOLOGIES M201 TEACHER'S GUIDE VOLUME 2 (U) SOFTECH INC WALTHAM MA 1986 DAAB07-83-C-K506

6/6

UNCLASSIFIED F/G 5/9 NL

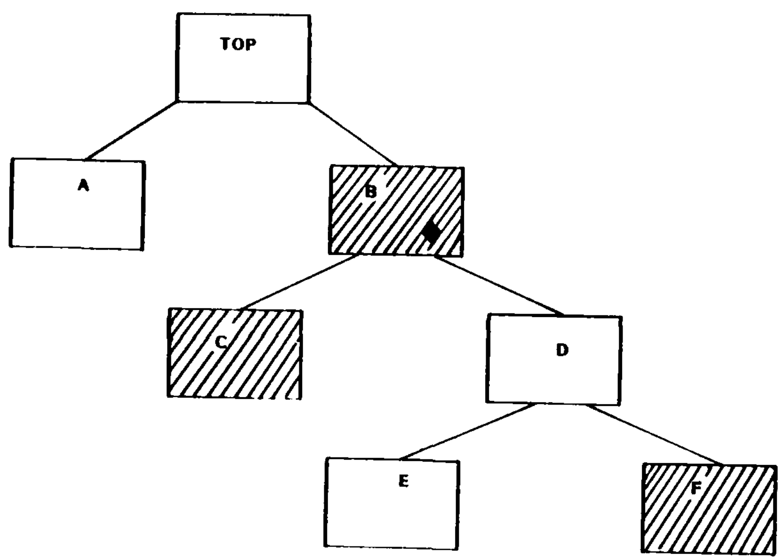


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

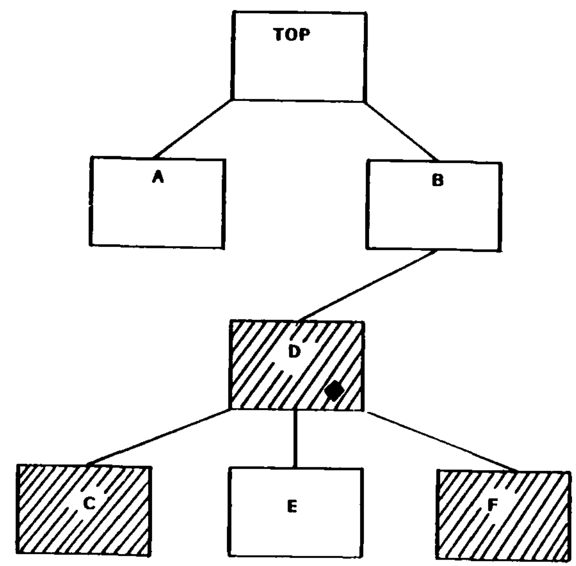
SCOPE OF EFFECT/SCOPE OF CONTROL

EXAMPLE

- AN ADEQUATE FIX ...



- IDEALLY, RESTRUCTURING IS REQUIRED ...



INSTRUCTOR NOTES

USE OF THE PREVIOUS TECHNIQUES WILL HELP IN ACHIEVING A GOOD CONTROL STRUCTURE.

CONTROL STRUCTURE

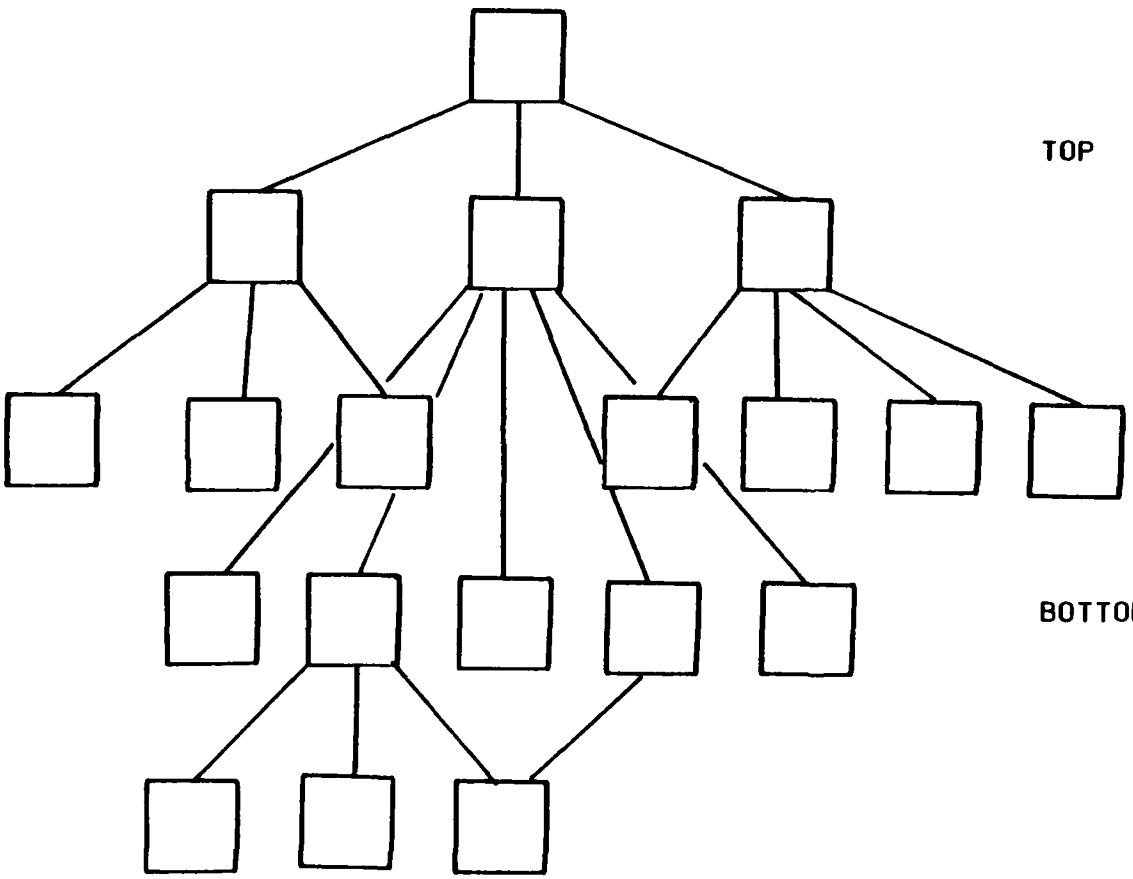
- CONTROL STRUCTURE IS A VIEW OF THE HIERARCHY WHICH ATTRIBUTES DIFFERENT CHARACTERISTICS TO THE TOP AND BOTTOM OF THE HIERARCHY.

- CONTROL STRUCTURE, IF ENFORCED WILL HELP MAKE THE RESULTING SYSTEM EASIER TO EXPAND AND MORE RELIABLE.

- PROPER CONTROL STRUCTURE IS ACHIEVED BY DELIBERATE RESTRUCTURING TO CONFORM TO CONTROL STRUCTURE GUIDELINES.

INSTRUCTOR NOTES

CONTROL STRUCTURE



TOP
- HIGHLY TRANSFORMED DATA
- EXECUTIVE MODULES
- MODERATE COUPLING
- MODERATE COHESION
- CALLED LESS OFTEN

BOTTOM
- RAW DATA
- WORKER MODULES
- LOW COUPLING
- HIGH COHESION
- CALLED MORE FREQUENTLY

INSTRUCTOR NOTES

LEFT DIAGRAM

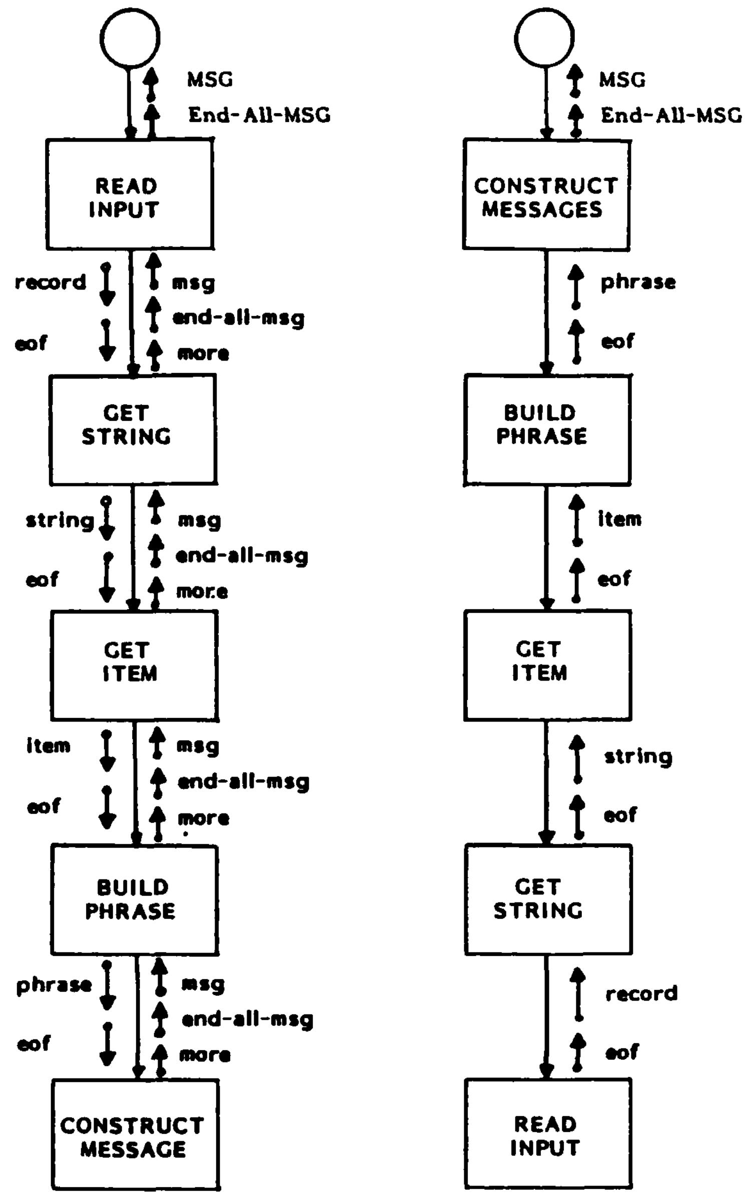
EACH TIME A SUBORDINATE NEEDS DATA HE SENDS BACK A FLAG TO HIS BOSS WHO SENDS BACK A FLAG UP THE CHAIN, ETC. THIS IS NOT A GOOD DESIGN, IT HAS POOR COUPLING.

RIGHT DIAGRAM

THIS IS THE INVERSE OF THE LEFT ONE. "CONSTRUCT MESSAGES" IS FUNCTIONALLY COHESIVE. MESSAGES ARE BUILT FROM PHRASES, ETC.

CONTROL STRUCTURE
EXAMPLE

- PHYSICAL TO LOGICAL DATA TRANSFORMS HAVE LESS COUPLING ...



INSTRUCTOR NOTES

DESIGN HEURISTICS

SUMMARY OF USAGE

- APPLY DESIGN HEURISTICS WITH FOLLOWING GOALS IN MIND:
  - HIGH FAN-IN } MOSQUE
  - MODERATE FAN-OUT } MOSQUE
  - SCOPE OF EFFECT <u>WITHIN</u> SCOPE OF CONTROL
  - MINIMAL CONTROL STRUCTURE
- HEURISTICS ARE MORE SUBJECTIVE THAN COUPLING AND COHESION METRICS

Material: Software Engr'g Methodologies (M201), Volume II

We would appreciate your comments on this material and would like you to complete this brief questionaire. The completed questionaire should be forwarded to the address on the back of this page. Thank you in advance for your time and effort.

1. Your name, company or affiliation, address and phone number.

2. Was the material accurate and technically correct?

   Yes ☐ No ☐

   Comments:

3. Were there any typographical errors?

   Yes ☐ No ☐

   If yes, on what pages?

4. Was the material organized and presented appropriately for your applications?

   Yes ☐ No ☐

   Comments:

5. General Comments:

place
stamp
here

COMMANDER
US ARMY MATERIEL COMMAND
ATTN: AMCDE-SB (OGLESBY)
5001 EISENHOWER AVENUE
ALEXANDRIA, VIRGINIA 22233

END
DTIC
FILMED
4-86